HW7

Avinash Ramu

October 30, 2016

Q16.3

The coefficient on time is -0.37, for each unit increase in time, the CD4 square root decreases by -0.37

```
#Use Andrew Stone's code from HW3 to ensure similar pre-processing
library(lme4)

## Loading required package: Matrix

library(arm)

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

##
## arm (Version 1.9-1, built: 2016-8-21)

## Working directory is /Users/conradlab/src/MultilevelModeling/HW/hw7

library(rjags); library(R2jags)

## Loading required package: coda

##
## Attaching package: 'coda'

## The following object is masked from 'package:arm':
##
##     traceplot

## Linked to JAGS 4.2.0

## Loaded modules: basemod,bugs

##
## Attaching package: 'R2jags'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot

## The following object is masked from 'package:arm':
##
##     traceplot

# Loading HIV data
hiv.data <- read.csv("allvar.csv", header=TRUE)
# Square root transformation of the CD4PCT
hiv.data$rootCD4 <- sqrt(hiv.data$CD4PCT)
# Creation of time variable
hiv.data$time <- hiv.data$visage - hiv.data$baseage
# Removing those cases that have NAs for the DV or the time variable
data.noNA.CD4 <- hiv.data[complete.cases(hiv.data[,4]),]
data.noNA.CD4 <- data.noNA.CD4[complete.cases(data.noNA.CD4[,11]),]
# Creating indicators for each of the children
inddummies <- as.factor(data.noNA.CD4$newpid)

mlm.1 <- lmer(rootCD4 ~ time + (1 | inddummies), data=data.noNA.CD4)
display(mlm.1)

## lmer(formula = rootCD4 ~ time + (1 | inddummies), data = data.noNA.CD4)
##             coef.est coef.se
## (Intercept)  4.76     0.10
## time        -0.37     0.05
##
## Error terms:
##  Groups     Name        Std.Dev.
##  inddummies (Intercept) 1.40
##  Residual               0.77
## ---
## number of obs: 1072, groups: inddummies, 250
## AIC = 3148.8, DIC = 3126.9
## deviance = 3133.9

n <- nrow(data.noNA.CD4)
y <- data.noNA.CD4$rootCD4
time <- data.noNA.CD4$time

# get ind index variable
ui <- unique(inddummies)
J <- length(ui)
ind <- rep (NA, J)
for (i in 1:J) {
  ind[inddummies == ui[i]] <- i
}


the.model <- "model{
```

```
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[ind[i]] + b * time[i]
  }
  b ~ dnorm (0, .0001)
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    a[j] ~ dnorm (mu.a, tau.a)
  }
  mu.a ~ dnorm (0, .0001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"
data <- list ("n", "J", "y", "time", "ind")
# Defining the initial values that your model's parameters (values you
*don't* already know)
inits <- function (){list(a=rnorm(J),
                          b=rnorm(1),
                          sigma.y=runif(1),
                          sigma.a=runif(1))}
# Defining which parameters of your model you want JAGS to return to you
# In the book (page 366), they are missing "g.0" and "g.1"
parameters <- c ("b", "sigma.y", "sigma.a")
# Now, we can actually run the model with the jags() function
m1 <- jags(data=data,
           inits=inits,
           parameters.to.save=parameters,
           model.file=textConnection(the.model), # Note the
textConnection() function
           n.chains=3,
           n.iter=5000,
           DIC=F)

## module glm loaded

## module dic loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1072
##    Unobserved stochastic nodes: 254
##    Total graph size: 4888
##
## Initializing model

m1
```

```
## Inference for Bugs model at "5", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##         mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b        -0.366   0.054 -0.470 -0.403 -0.366 -0.328 -0.260 1.001  2500
## sigma.a   1.405   0.068  1.280  1.357  1.403  1.450  1.543 1.001  3300
## sigma.y   0.774   0.019  0.739  0.760  0.773  0.786  0.812 1.001  3800
##
## 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).
```

PartB

The coefficient on time is similar as part A, the coefficient on treatment is 0.180 and the coefficient on baseage is -0.119. The Rhat on the coefficient for treatment appears to be higher.

```
treatment <- data.noNA.CD4$treatmnt
baseage <- data.noNA.CD4$baseage
the.model2 <- "model{
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[ind[i]] + b1 * time[i] + b2 * treatment[i] + b3 *
baseage[i]
  }
  b1 ~ dnorm (0, .0001)
  b2 ~ dnorm (0, .0001)
  b3 ~ dnorm (0, .0001)
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    a[j] ~ dnorm (mu.a, tau.a)
  }
  mu.a ~ dnorm (0, .0001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"
data <- list ("n", "J", "y", "time", "ind", "treatment", "baseage")
inits <- function (){list(a=rnorm(J),
                                b1 = rnorm(1),
                                b2 = rnorm(1),
                                b3 = rnorm(1),
                                sigma.y=runif(1),
                                sigma.a=runif(1))}
parameters <- c ("b1", "b2", "b3", "sigma.y", "sigma.a")
m2 <- jags(data=data,
               inits=inits,
               parameters.to.save=parameters,
               model.file=textConnection(the.model2), # Note the
```

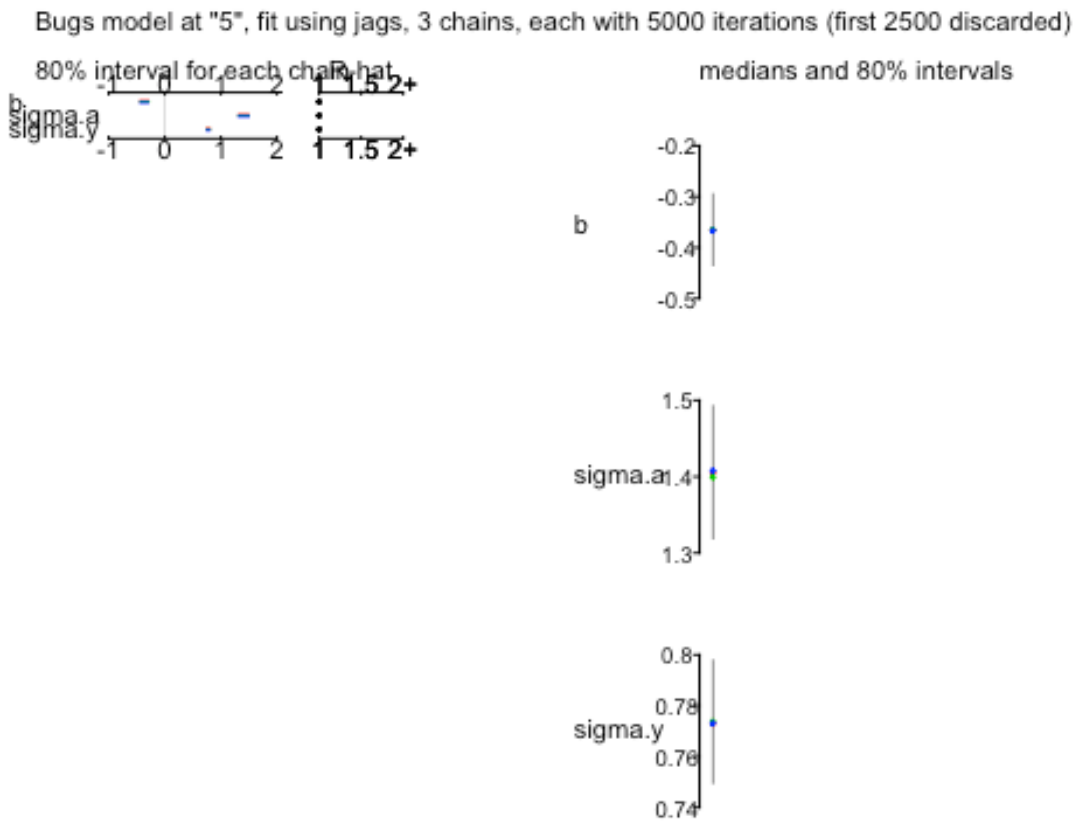
```
textConnection() function
                n.chains=3,
                n.iter=5000,
                DIC=F)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1072
##    Unobserved stochastic nodes: 256
##    Total graph size: 7279
##
## Initializing model

m2

## Inference for Bugs model at "6", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##         mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b1       -0.363   0.054 -0.469 -0.399 -0.363 -0.327 -0.257 1.001  3800
## b2        0.187   0.180 -0.170  0.069  0.186  0.306  0.543 1.004   610
## b3       -0.119   0.040 -0.198 -0.146 -0.118 -0.092 -0.039 1.002  1600
## sigma.a   1.384   0.070  1.255  1.335  1.382  1.429  1.529 1.001  3800
## sigma.y   0.774   0.019  0.738  0.761  0.774  0.787  0.812 1.002  1600
##
## 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).
```

Part C - written

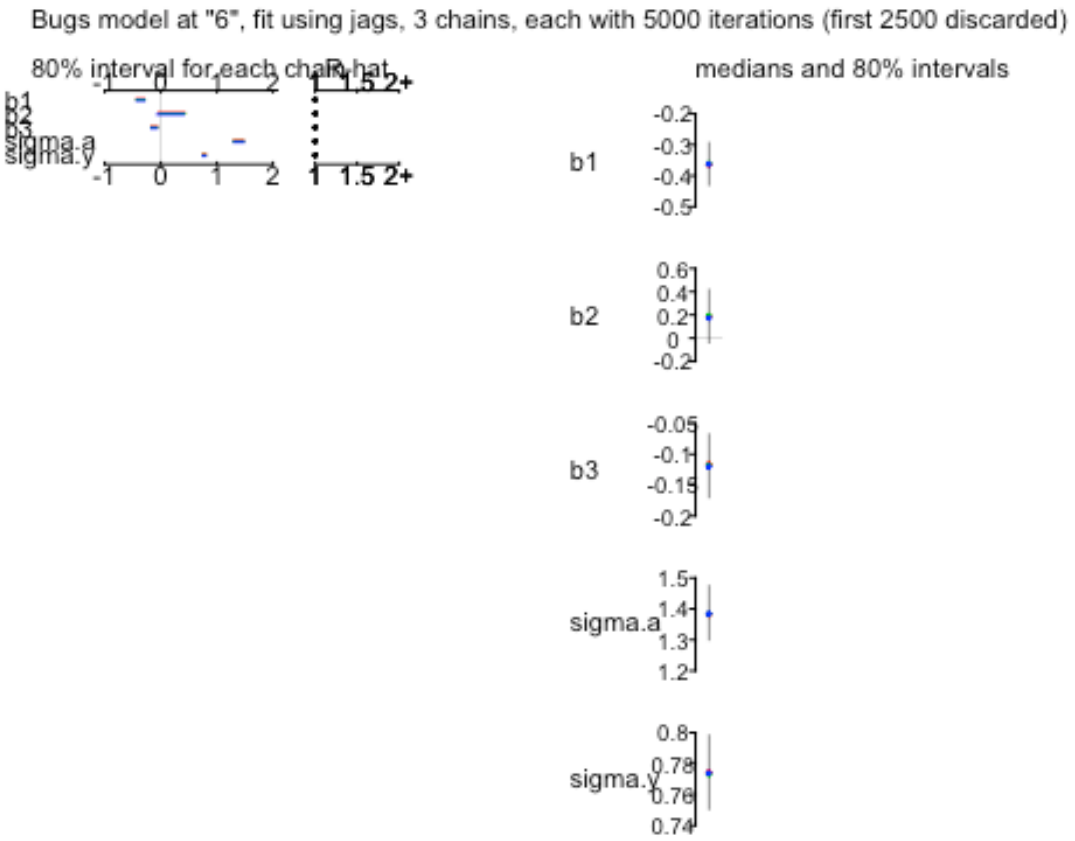
Part D

Plot the models and explain

```
plot(m1)
```


Bugs model at "5", fit using jags, 3 chains, each with 5000 iterations (first 2500 discarded)
80% interval for each chain
R-hat
medians and 80% intervals
b
sigma.a
sigma.y
-1
0
1
2
1
1.5
2+
-0.2
-0.3
-0.4
-0.5
1.5
1.4
1.3
0.8
0.78
0.76
0.74

```
plot(m2)
```




```
##Use Andy's tutorial code

# Setting up the data (from Gelman & Hill's website)
srrs2 <- read.table ("srrs2.dat", header=T, sep=",")
mn <- srrs2$state=="MN"
radon <- srrs2$activity[mn]
log.radon <- log (ifelse (radon==0, .1, radon))
floor <- srrs2$floor[mn]       # 0 for basement, 1 for first floor
n <- length(radon)
y <- log.radon
x <- floor

# get county index variable
county.name <- as.vector(srrs2$county[mn])
uniq <- unique(county.name)
J <- length(uniq)
county <- rep (NA, J)
for (i in 1:J){
  county[county.name==uniq[i]] <- i
}

srrs2.fips <- srrs2$stfips*1000 + srrs2$cntyfips
cty <- read.table ("cty.dat", header=T, sep=",")
```

```
usa.fips <- 1000*cty[,"stfips"] + cty[,"ctfips"]
usa.rows <- match (unique(srrs2.fips[mn]), usa.fips)
uranium <- cty[usa.rows,"Uppm"]
u <- log (uranium)
```

Setting up JAGS

A - normal prior distributions with mean 5 and standard deviation 1000 to the coefficients for floor and uranium

```
the.model <- "model{
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[county[i]] + b*x[i]
  }
  b ~ dnorm (5, .000001)
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    a[j] ~ dnorm (a.hat[j], tau.a)
    a.hat[j] <- g.0 + g.1*u[j]
  }
  g.0 ~ dnorm (0, .0001)
  g.1 ~ dnorm (5, .000001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"

# Defining the data you will pass into the model -- you *already know* these
values
radon.data <- list ("n", "J", "x", "y", "county", "u")

# Defining the initial values that your model's parameters (values you
*don't* already know)
radon.inits <- function (){list(a=rnorm(J),
                                b=rnorm(1),
                                g.0=rnorm(1),
                                g.1=rnorm(1),
                                sigma.y=runif(1),
                                sigma.a=runif(1))}

# Defining which parameters of your model you want JAGS to return to you
# In the book (page 366), they are missing "g.0" and "g.1"
radon.parameters <- c ("b", "sigma.y", "sigma.a", "g.0", "g.1")

# Now, we can actually run the model with the jags() function
radon.3 <- jags(data=radon.data,
```

```
                inits=radon.inits,
                parameters.to.save=radon.parameters,
                model.file=textConnection(the.model), # Note the
textConnection() function
                n.chains=3,
                n.iter=5000,
                DIC=F)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 919
##    Unobserved stochastic nodes: 90
##    Total graph size: 3266
##
## Initializing model

radon.3

## Inference for Bugs model at "7", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##          mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b         -0.667   0.069 -0.802 -0.714 -0.668 -0.621 -0.533 1.001  3800
## g.0        1.464   0.039  1.389  1.438  1.463  1.489  1.540 1.003   820
## g.1        0.713   0.093  0.524  0.652  0.717  0.775  0.890 1.006   430
## sigma.a    0.163   0.051  0.066  0.128  0.162  0.196  0.268 1.031   160
## sigma.y    0.760   0.019  0.725  0.747  0.760  0.772  0.799 1.003   820
##
## 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).
```

B - normal prior distributions with mean 0 and standard deviation 0.1 to the coefficients for floor and uranium

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

  for (j in 1:J){
    a[j] ~ dnorm (a.hat[j], tau.a)
    a.hat[j] <- g.0 + g.1*u[j]
  }
  g.0 ~ dnorm (0, .0001)
  g.1 ~ dnorm (0, 100)
```

```
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"

# Defining the data you will pass into the model -- you *already know* these
values
radon.data <- list ("n", "J", "x", "y", "county", "u")

# Defining the initial values that your model's parameters (values you
*don't* already know)
radon.inits <- function (){list(a=rnorm(J),
                                b=rnorm(1),
                                g.0=rnorm(1),
                                g.1=rnorm(1),
                                sigma.y=runif(1),
                                sigma.a=runif(1))}

# Defining which parameters of your model you want JAGS to return to you
# In the book (page 366), they are missing "g.0" and "g.1"
radon.parameters <- c ("b", "sigma.y", "sigma.a", "g.0", "g.1")

# Now, we can actually run the model with the jags() function
radon.3 <- jags(data=radon.data,
                inits=radon.inits,
                parameters.to.save=radon.parameters,
                model.file=textConnection(the.model), # Note the
textConnection() function
                n.chains=3,
                n.iter=5000,
                DIC=F)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 919
##    Unobserved stochastic nodes: 90
##    Total graph size: 3266
##
## Initializing model

radon.3

## Inference for Bugs model at "8", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##         mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b        -0.455   0.058 -0.568 -0.494 -0.456 -0.416 -0.338 1.001  3800
## g.0       1.413   0.043  1.330  1.384  1.413  1.440  1.498 1.002  1700
## g.1       0.338   0.086  0.167  0.280  0.337  0.395  0.512 1.001  3800
```

```
## sigma.a   0.223   0.051  0.125  0.188  0.222  0.257  0.325 1.001  2700
## sigma.y   0.763   0.019  0.727  0.750  0.763  0.776  0.801 1.001  3800
##
## 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).
```

C - normal prior distributions with mean 5 and standard deviation 1 to the coefficients for floor and uranium

```
the.model <- "model{
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[county[i]] + b*x[i]
  }
  b ~ dnorm (5, 1)
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    a[j] ~ dnorm (a.hat[j], tau.a)
    a.hat[j] <- g.0 + g.1*u[j]
  }
  g.0 ~ dnorm (0, .0001)
  g.1 ~ dnorm (5, 1)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"

# Defining the data you will pass into the model -- you *already know* these
values
radon.data <- list ("n", "J", "x", "y", "county", "u")

# Defining the initial values that your model's parameters (values you
*don't* already know)
radon.inits <- function (){list(a=rnorm(J),
                                b=rnorm(1),
                                g.0=rnorm(1),
                                g.1=rnorm(1),
                                sigma.y=runif(1),
                                sigma.a=runif(1))}

# Defining which parameters of your model you want JAGS to return to you
# In the book (page 366), they are missing "g.0" and "g.1"
radon.parameters <- c ("b", "sigma.y", "sigma.a", "g.0", "g.1")

# Now, we can actually run the model with the jags() function
radon.3 <- jags(data=radon.data,
                inits=radon.inits,
                parameters.to.save=radon.parameters,
                model.file=textConnection(the.model), # Note the
```

```
textConnection() function
                n.chains=3,
                n.iter=5000,
                DIC=F)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 919
##    Unobserved stochastic nodes: 90
##    Total graph size: 3266
##
## Initializing model

radon.3

## Inference for Bugs model at "9", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##         mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b        -0.640   0.068 -0.770 -0.686 -0.642 -0.594 -0.509 1.003   810
## g.0       1.462   0.039  1.387  1.435  1.462  1.487  1.539 1.005   460
## g.1       0.762   0.094  0.576  0.698  0.764  0.825  0.945 1.004   550
## sigma.a   0.154   0.050  0.054  0.121  0.154  0.188  0.254 1.014  2800
## sigma.y   0.761   0.019  0.725  0.748  0.761  0.774  0.799 1.001  3800
##
## 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).
```

D - t prior distributions with mean 5, standard deviation 1, and 4 degrees of freedom.

```
the.model <- "model{
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[county[i]] + b*x[i]
  }
  b ~ dt (5, 1, 4)
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J){
    a[j] ~ dnorm (a.hat[j], tau.a)
    a.hat[j] <- g.0 + g.1*u[j]
  }
  g.0 ~ dnorm (0, .0001)
  g.1 ~ dt (5, 1, 4)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"
```

```r
# Defining the data you will pass into the model -- you *already know* these
values
radon.data <- list ("n", "J", "x", "y", "county", "u")

# Defining the initial values that your model's parameters (values you
*don't* already know)
radon.inits <- function (){list(a=rnorm(J),
                                b=rnorm(1),
                                g.0=rnorm(1),
                                g.1=rnorm(1),
                                sigma.y=runif(1),
                                sigma.a=runif(1))}

# Defining which parameters of your model you want JAGS to return to you
# In the book (page 366), they are missing "g.0" and "g.1"
radon.parameters <- c ("b", "sigma.y", "sigma.a", "g.0", "g.1")

# Now, we can actually run the model with the jags() function
radon.3 <- jags(data=radon.data,
                inits=radon.inits,
                parameters.to.save=radon.parameters,
                model.file=textConnection(the.model), # Note the
textConnection() function
                n.chains=3,
                n.iter=5000,
                DIC=F)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 919
##    Unobserved stochastic nodes: 90
##    Total graph size: 3268
##
## Initializing model
```

```r
radon.3
```

```
## Inference for Bugs model at "10", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##          mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b         -0.663   0.070 -0.798 -0.710 -0.663 -0.614 -0.528 1.003   770
## g.0        1.463   0.040  1.385  1.437  1.465  1.489  1.542 1.006   480
## g.1        0.730   0.094  0.540  0.668  0.734  0.794  0.905 1.008   260
## sigma.a    0.162   0.047  0.075  0.130  0.161  0.192  0.263 1.022   170
## sigma.y    0.760   0.018  0.725  0.747  0.760  0.772  0.796 1.001  3800
##
```

```
## 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).
```

E1 - try Uniform(−100,100) prior distributions

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

  for (j in 1:J){
    a[j] ~ dnorm (a.hat[j], tau.a)
    a.hat[j] <- g.0 + g.1*u[j]
  }
  g.0 ~ dnorm (0, .0001)
  g.1 ~ dunif (-100, 100)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"

# Defining the data you will pass into the model -- you *already know* these
values
radon.data <- list ("n", "J", "x", "y", "county", "u")

# Defining the initial values that your model's parameters (values you
*don't* already know)
radon.inits <- function (){list(a=rnorm(J),
                                b=rnorm(1),
                                g.0=rnorm(1),
                                g.1=rnorm(1),
                                sigma.y=runif(1),
                                sigma.a=runif(1))}

# Defining which parameters of your model you want JAGS to return to you
# In the book (page 366), they are missing "g.0" and "g.1"
radon.parameters <- c ("b", "sigma.y", "sigma.a", "g.0", "g.1")

# Now, we can actually run the model with the jags() function
radon.3 <- jags(data=radon.data,
                inits=radon.inits,
                parameters.to.save=radon.parameters,
                model.file=textConnection(the.model), # Note the
textConnection() function
                n.chains=3,
                n.iter=5000,
                DIC=F)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 919
##    Unobserved stochastic nodes: 90
##    Total graph size: 3267
##
## Initializing model
```

```
radon.3
```

```
## Inference for Bugs model at "7", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##         mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b        -0.668   0.070 -0.805 -0.717 -0.669 -0.621 -0.531 1.002  1600
## g.0       1.466   0.040  1.392  1.438  1.465  1.492  1.543 1.003   740
## g.1       0.718   0.097  0.529  0.654  0.720  0.785  0.909 1.001  2400
## sigma.a   0.163   0.049  0.074  0.126  0.162  0.196  0.264 1.018   150
## sigma.y   0.759   0.018  0.724  0.747  0.759  0.771  0.796 1.001  3800
##
## 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).
```

E2 - try Uniform(−1,1) prior distributions

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

  for (j in 1:J){
    a[j] ~ dnorm (a.hat[j], tau.a)
    a.hat[j] <- g.0 + g.1*u[j]
  }
  g.0 ~ dnorm (0, .0001)
  g.1 ~ dunif (-1, 1)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"

# Defining the data you will pass into the model -- you *already know* these
values
radon.data <- list ("n", "J", "x", "y", "county", "u")

# Defining the initial values that your model's parameters (values you
```

```
*don't* already know)
radon.inits <- function (){list(a=rnorm(J),
                                b= 0,
                                g.0=rnorm(1),
                                g.1 = 0,
                                sigma.y=runif(1),
                                sigma.a=runif(1))}

# Defining which parameters of your model you want JAGS to return to you
# In the book (page 366), they are missing "g.0" and "g.1"
radon.parameters <- c ("b", "sigma.y", "sigma.a", "g.0", "g.1")

# Now, we can actually run the model with the jags() function
radon.3 <- jags(data=radon.data,
                inits=radon.inits,
                parameters.to.save=radon.parameters,
                model.file=textConnection(the.model), # Note the
textConnection() function
                n.chains=3,
                n.iter=5000,
                DIC=F)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 919
##    Unobserved stochastic nodes: 90
##    Total graph size: 3267
##
## Initializing model

radon.3

## Inference for Bugs model at "8", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##         mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b        -0.670   0.068 -0.802 -0.716 -0.671 -0.623 -0.537 1.002  2200
## g.0       1.467   0.038  1.394  1.441  1.466  1.493  1.544 1.002  1800
## g.1       0.722   0.091  0.542  0.661  0.724  0.784  0.894 1.001  3800
## sigma.a   0.157   0.047  0.068  0.126  0.156  0.187  0.252 1.024   350
## sigma.y   0.760   0.018  0.725  0.748  0.759  0.772  0.796 1.002  1600
##
## 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).
```

Rerun Q16.3 using 17.2 and 17.3

Use varying slopes on model fit in 16.3(B)

I used a varying slope on the time variable here. There are fixed slopes for treatment, time and age as in the model in 12.2(B). There is a varying intercept term as well.

```
y <- data.noNA.CD4$rootCD4
time <- data.noNA.CD4$time
treatment <- data.noNA.CD4$treatmnt
baseage <- data.noNA.CD4$baseage
inddummies <- as.factor(data.noNA.CD4$newpid)
n <- nrow(data.noNA.CD4)
# get ind index variable
ui <- unique(inddummies)
J <- length(ui)
ind <- rep (NA, J)
for (i in 1:J) {
  ind[inddummies == ui[i]] <- i
}
the.model3 <- "model{
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[ind[i]] + b1 * time[i] + b2 * treatment[i] + b3 *
baseage[i] + b4[ind[i]] * time[i]
  }
  b1 ~ dnorm (0, .0001)
  b2 ~ dnorm (0, .0001)
  b3 ~ dnorm (0, .0001)
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)
  for (j in 1:J){
    a[j] ~ dnorm (a.hat[j], tau.a)
    b4[j] ~ dnorm (b.hat4[j], tau.b4)
    a.hat[j] <- mu.a
    b.hat4[j] <- mu.b4
  }
  mu.a ~ dnorm (0, .0001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
  mu.b4 ~ dnorm (0, .0001)
  tau.b4 <- pow(sigma.b4, -2)
  sigma.b4 ~ dunif (0, 100)
}"
data <- list ("n", "J", "y", "time", "ind", "treatment", "baseage")
inits <- function (){list(a=rnorm(J),
                          b1 = rnorm(1),
                          b2 = rnorm(1),
                          b3 = rnorm(1),
                          b4 = rnorm(J),
```

```
                                sigma.y=runif(1),
                                sigma.a=runif(1),
                                sigma.b4 = runif(1))}
parameters <- c ("b1", "b2", "b3", "sigma.y", "sigma.a")
m3 <- jags(data=data,
               inits=inits,
               parameters.to.save=parameters,
               model.file=textConnection(the.model3), # Note the
textConnection() function
               n.chains=3,
               n.iter=5000,
               DIC=F)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1072
##    Unobserved stochastic nodes: 508
##    Total graph size: 8609
##
## Initializing model

m3

## Inference for Bugs model at "11", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##         mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## b1        0.189   1.735 -2.853 -0.963 -0.344  1.273  4.157 3.075     4
## b2        0.163   0.180 -0.192  0.043  0.163  0.282  0.517 1.007   460
## b3       -0.125   0.040 -0.204 -0.152 -0.124 -0.097 -0.047 1.001  3200
## sigma.a   1.359   0.069  1.232  1.312  1.355  1.403  1.505 1.001  2900
## sigma.y   0.720   0.020  0.681  0.707  0.720  0.733  0.759 1.002  1100
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
```

Q17.5

```
##From Andrew Stone's HW4 solution
age.data <- read.csv("age.guessing.csv")
# Empty matrix to put observations into
analysis.matrix <- matrix(NA, nrow=100, ncol=3)
ages <- c()
group <- c()
person <- rep(c(1:10), times=10) # Creating individual ID variable
# For loop creates the (non-abs value) dependent variable and the group ID
variable
for(i in 1:10){
ages <- c(ages, as.integer(age.data[i,3:12]))
```

```
group <- c(group, rep(age.data[i,1], times=10))
}
# Adding the variables to the matrix
analysis.matrix[,1] <- ages
analysis.matrix[,2] <- group
analysis.matrix[,3] <- person
# Turning the matrix into a data frame
model.data <- data.frame(analysis.matrix)
# Giving the variables in the data frame names
colnames(model.data) <- c("error", "group.id", "person.id")
# Turning the group and individual ID variables into factors
model.data$group.id <- as.factor(model.data$group.id)
model.data$person.id <- as.factor(model.data$person.id)
# Making the true DV by taking the absolute value
model.data$error <- abs(model.data$error)
# Multilevel model with separate coefficinets for each group and individual
age.model <- lmer(error ~ 1 + (1 | group.id) + (1 | person.id),
data=model.data)
summary(age.model)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: error ~ 1 + (1 | group.id) + (1 | person.id)
##    Data: model.data
##
## REML criterion at convergence: 545.5
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.8256 -0.5603 -0.1148  0.6566  3.8882
##
## Random effects:
##  Groups    Name        Variance Std.Dev.
##  group.id  (Intercept)  0.2002  0.4475
##  person.id (Intercept) 10.9625  3.3110
##  Residual              10.9320  3.3064
## Number of obs: 100, groups:  group.id, 10; person.id, 10
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)    5.470      1.107   4.941
```

```
n <- nrow(model.data)
y <- model.data$error
group.id <- model.data$group.id
person.id <- model.data$person.id
up <- unique(person.id)
J1 <- length(up)
person <- rep (NA, J1)
for (i in 1:J1) {
  person[person.id == up[i]] <- i
```

```
}
ug <- unique(group.id)
J2 <- length(ug)
group <- rep (NA, J2)
for (i in 1:J2) {
  group[group.id == ug[i]] <- i
}

the.model4 <- "model{
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a + b[person[i]] + c[group[i]]
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)
  for (j in 1:J1){
    b[j] ~ dnorm (mu.b, tau.b[j])
    tau.b[j] <- pow(sigma.b[j], -2)
    sigma.b[j] ~ dunif (0, 100)
  }
  for (j in 1:J2){
    c[j] ~ dnorm (mu.c, tau.c[j])
    tau.c[j] <- pow(sigma.c[j], -2)
    sigma.c[j] ~ dunif (0, 100)
  }
  a ~ dnorm (mu.a, tau.a)
  mu.a ~ dnorm (0, .0001)
  mu.b ~ dnorm (0, .0001)
  mu.c ~ dnorm (0, .0001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}"
data <- list ("n", "J1", "J2", "y", "person", "group")
inits <- function (){list(a = rnorm(1),
                          b = rnorm(J1),
                          c = rnorm(J2),
                          sigma.y = runif(1),
                          sigma.a = runif(1),
                          sigma.b = runif(J1),
                          sigma.c = runif(J2))}
parameters <- c ("b", "c", "sigma.y", "sigma.a", "sigma.b", "sigma.c")
m4 <- jags(data=data,
           inits=inits,
           parameters.to.save=parameters,
           model.file=textConnection(the.model4), # Note the
textConnection() function
           n.chains=3,
           n.iter=5000,
           DIC=F)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 46
##    Total graph size: 544
##
## Initializing model

m4

## Inference for Bugs model at "12", fit using jags,
##  3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 2
##  n.sims = 3750 iterations saved
##             mu.vect sd.vect     2.5%      25%     50%    75%   97.5%  Rhat
## b[1]          9.487  88.425 -151.887  -48.150  10.923 74.733 173.829 2.922
## b[2]          2.921  88.429 -158.054  -54.764   3.964 67.996 167.172 2.921
## b[3]          3.279  88.425 -157.684  -54.166   4.518 68.786 167.610 2.921
## b[4]          5.493  88.413 -155.357  -52.026   6.823 70.426 169.816 2.921
## b[5]          1.924  88.439 -158.549  -55.756   3.194 67.013 165.881 2.921
## b[6]          5.210  88.445 -155.905  -52.416   6.584 70.399 169.627 2.922
## b[7]         12.094  88.425 -149.033  -45.418  13.602 77.003 176.166 2.920
## b[8]          1.768  88.434 -158.798  -56.048   2.765 66.697 165.903 2.921
## b[9]          6.396  88.466 -154.381  -51.383   7.641 71.289 170.493 2.921
## b[10]         2.718  88.450 -158.260  -55.257   3.673 67.512 166.820 2.920
## c[1]        -59.882  67.623 -160.135 -122.091 -57.098  5.601  48.690 5.428
## c[2]        -63.328  67.610 -163.831 -125.448 -60.438  1.765  45.221 5.426
## c[3]        -60.815  67.646 -161.107 -123.062 -57.836  4.664  47.727 5.421
## c[4]        -61.063  67.615 -161.493 -123.473 -58.180  4.404  47.374 5.433
## c[5]        -61.518  67.614 -162.069 -123.695 -58.686  3.912  47.025 5.427
## c[6]        -61.906  67.585 -162.352 -124.342 -58.996  3.471  46.207 5.422
## c[7]        -60.539  67.610 -161.117 -122.673 -57.402  5.099  47.757 5.432
## c[8]        -61.974  67.621 -162.440 -124.101 -59.007  3.179  46.422 5.415
## c[9]        -62.888  67.614 -163.426 -125.208 -59.860  2.199  45.717 5.423
## c[10]       -61.667  67.604 -162.080 -124.240 -58.410  3.536  46.701 5.422
## sigma.a      49.927  28.711    2.684   24.996  50.628 74.802  97.264 1.021
## sigma.b[1]   30.573  26.354    1.587    9.315  21.460 46.033  93.068 1.001
## sigma.b[2]   23.972  25.804    0.483    4.329  12.814 35.868  91.214 1.001
## sigma.b[3]   24.000  26.109    0.336    3.862  12.909 36.530  91.180 1.013
## sigma.b[4]   23.957  25.450    0.521    4.727  13.360 36.225  91.097 1.001
## sigma.b[5]   25.700  26.745    0.373    4.598  14.514 39.987  93.297 1.009
## sigma.b[6]   24.451  25.583    0.484    4.638  13.905 37.270  90.387 1.001
## sigma.b[7]   33.646  25.978    3.583   12.542  25.578 49.672  94.194 1.003
## sigma.b[8]   25.043  25.912    0.560    4.814  14.508 38.774  90.991 1.002
## sigma.b[9]   25.943  26.124    0.652    5.355  15.334 40.775  91.183 1.001
## sigma.b[10]  23.251  25.702    0.258    3.804  12.142 35.159  89.917 1.003
## sigma.c[1]   23.029  25.226    0.336    3.845  12.090 35.359  89.383 1.003
## sigma.c[2]   22.101  24.686    0.449    3.792  11.687 32.284  88.568 1.002
## sigma.c[3]   21.704  25.072    0.318    3.184  10.545 31.288  89.664 1.023
```

```
## sigma.c[4]   20.840  24.342    0.190    2.896  10.063 30.940  87.389 1.003
## sigma.c[5]   20.781  24.982    0.197    2.501   9.212 31.283  87.886 1.008
## sigma.c[6]   21.533  25.768    0.213    2.462   9.669 31.954  90.367 1.002
## sigma.c[7]   22.352  25.390    0.246    3.347  11.362 33.421  89.897 1.007
## sigma.c[8]   22.523  25.817    0.269    3.027  10.903 34.582  89.545 1.001
## sigma.c[9]   21.402  24.903    0.249    3.287  10.311 31.426  88.733 1.003
## sigma.c[10]  21.440  25.623    0.204    2.541   9.681 31.568  90.411 1.004
## sigma.y       3.348   0.267    2.873    3.162   3.331  3.516   3.920 1.003
##             n.eff
## b[1]            4
## b[2]            4
## b[3]            4
## b[4]            4
## b[5]            4
## b[6]            4
## b[7]            4
## b[8]            4
## b[9]            4
## b[10]           4
## c[1]            3
## c[2]            3
## c[3]            3
## c[4]            3
## c[5]            3
## c[6]            3
## c[7]            3
## c[8]            3
## c[9]            3
## c[10]           3
## sigma.a       170
## sigma.b[1]   3800
## sigma.b[2]   3800
## sigma.b[3]    710
## sigma.b[4]   3300
## sigma.b[5]    450
## sigma.b[6]   2500
## sigma.b[7]   1500
## sigma.b[8]   1700
## sigma.b[9]   3800
## sigma.b[10]  1000
## sigma.c[1]    850
## sigma.c[2]   1700
## sigma.c[3]     96
## sigma.c[4]   2500
## sigma.c[5]    790
## sigma.c[6]   1200
## sigma.c[7]    400
## sigma.c[8]   3800
## sigma.c[9]    740
## sigma.c[10]   690
```

```
## sigma.y       990
##
## 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).
```