Multilevel Modeling

Answer Key: Homework 7

Andy Stone

November 10, 2016

16.3

a.

We recall that this model predicts `rootCD4` as a function of time with varying intercepts across children.

We can write this varying-intercepts model in JAGS with the following code, letting `time` $= x$ and `rootCD4` $= y$:

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[child[i]] + b*x[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)
}
```

To run the model, we first need to load and manipulate the data. I first load the data, create the dependent variable and the time variable, and remove those cases that have NA values for either of these variables.

```
library(foreign); library(arm); library(car); library(rjags); library(R2jags)
# Set seed for replication
set.seed(100)

# Loading and setting up the HIV data

# Loading HIV data
hiv <- read.csv("allvar.csv", header=TRUE)
# Square root transformation of the CD4PCT
hiv$rootCD4 <- sqrt(hiv$CD4PCT)
# Creation of time variable
hiv$time <- hiv$visage - hiv$baseage

# Removing those cases that have NAs for the DV or the time variable
data.noNA.CD4 <- hiv[complete.cases(hiv[,4]),]
data.noNA.CD4 <- data.noNA.CD4[complete.cases(data.noNA.CD4[,11]),]
```

Then, I move to defining the variables we will need to run our JAGS model. `n` is simply the number of unique observations, `J` is the number of unique children, `y` is the dependent variable, `x` is the time variable, and `child` is an indicator denoting which child an observation belongs to.

```
# Creating n
n <- length(data.noNA.CD4$rootCD4)

# Creating J
unique.child <- unique(data.noNA.CD4$newpid)
J <- length(unique.child)

# Creating y
y <- data.noNA.CD4$rootCD4

# Creating x
x <- data.noNA.CD4$time

# Creating child indicator
child <- rep(NA, J)
for (i in 1:J){
  child[data.noNA.CD4$newpid == unique.child[i]] <- i
}
```

Now, I move to setting up the data, initial values, and parameters we will need to pass to the `jags()` function to run the JAGS model. Then, I define the model itself. I save this to an object, rather than in a separate `.bug` file.

```
# Lists the data that will be contained in our JAGS model
hiv.data <- list("n","J","y","child","x")

# Function to return list of starting values of algorithm
# For example, rnorm(J) is vector of length J of random numbers from the N(0,1) distribution
# These are assigned to alpha to start the JAGS iterations
# We also do this for betas, mu.a's, sigma.y's, and sigma.a's -- that is, all parameters
hiv.inits <- function(){
  list(a=rnorm(J), b=rnorm(1), mu.a=rnorm(1),
       sigma.y=runif(1), sigma.a=runif(1))}

# This is vector of names of parameters we want to save from the JAGS run
hiv.parameters <- c("a", "b", "mu.a", "sigma.y", "sigma.a")

# The model itself

hiv.mod <- 'model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[child[i]] + b*x[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)
}'
```

Now, we are finally ready to run JAGS. I specify that our algorithm should use 3 chains to simulate parameter values, as well as specifying 500 iterations, with the standard burn-in period of the first half of the iterations.

```
# This is the actual JAGS run
hiv.1 <- jags(data=hiv.data, inits=hiv.inits, parameters.to.save=hiv.parameters,
              model.file=textConnection(hiv.mod), n.chains=3, n.iter=500, DIC=F)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 1072
   Unobserved stochastic nodes: 254
   Total graph size: 4888

Initializing model
```

I report an abbreviated version of the model output in Table 1 below. In interpreting the model, we see that the estimates on the group-level parameters vary – this is to be expected, as we expect that each child will have a different starting point for their `rootCD4` value. The estimate of the average intercept is 4.763, and is precisely estimated, with a standard error of 0.096. The standard error in the child-specific intercepts are larger than on the average intercepts. The estimate on the time parameter is -0.366, and is very precise with a standard error of 0.056. This suggests that, as time progresses, `rootCD4` tends to decrease. We see that the estimated within-child standard deviation $\hat{\sigma_y} = 0.77$ is estimated to be smaller than the across-child standard deviation of $\hat{\sigma_\alpha} = 1.40$. This makes sense – we expect less variation within a child on their CD4 levels than across children.

Looking at the $\hat{R}$ column, we see suggestive evidence that the algorithm converged – none of these values are above 1.01. Looking at the n_{eff} column suggests that we have enough effective simulations to achieve precise estimates of our parameters.

Table 1: Varying Intercepts Model

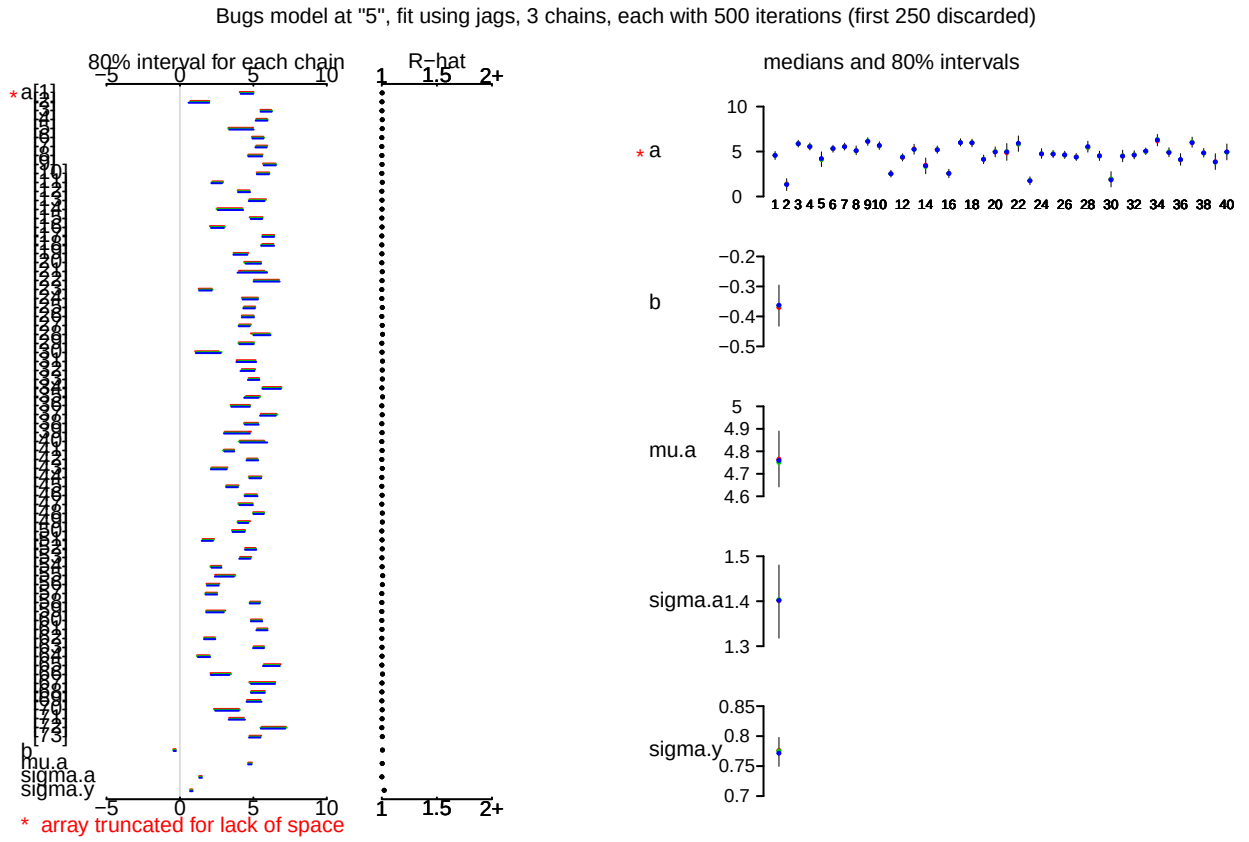
Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
α_1	4.562	0.334	3.923	4.338	4.554	4.785	5.238	1.006	610
α_2	1.355	0.511	0.384	1.022	1.371	1.683	2.352	1.009	260
⋮									
α_{250}	3.878	0.510	2.946	3.533	3.876	4.191	4.912	1.000	750
β	-0.366	0.056	-0.472	-0.402	-0.369	-0.328	-0.253	1.001	750
μ_α	4.763	0.096	4.579	4.698	4.760	4.828	4.944	1.003	750
σ_α	1.400	0.076	1.253	1.347	1.396	1.451	1.551	1.007	750
σ_y	0.774	0.020	0.737	0.760	0.772	0.787	0.815	1.007	420

DIC info (using the rule, pD = var(deviance)/2): pD = 340.2 and DIC = 2830.4

A second method we can use to aid in interpretation of model results is to examine the built-in graphical summary of a JAGS simulation, accessed through the `plot` function. The plot corroborates our findings presented above. The left-hand side provides estimates of the (first 73 or so) child-level parameters, with 80% uncertainty intervals, for each of the 3 chains. We also see the $\hat{R}$ values plotted for each of these parameter

estimates. On the right-hand side, the median estimates of the parameters in the model are plotted with 80% uncertainty bars.

```
plot(hiv.1)
```



b.

Now, I proceed to use JAGS to fit the model in Exercise 12.2(b). This model, we recall, adds two things to the previous model: group-level predictors for treatment and age at baseline. All that changes in the model, then, is how we model the group-level intercepts α_j. The new JAGS model is shown below, letting `treatment` $= t$ and `baseage` $= ba$:

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[child[i]] + b*x[i]
  }
  b ~ 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)
    a.hat[j] <- g0 + g1*t[j] + g2*ba[j]
```

```
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dnorm(0, 0.0001)
  g2 ~ dnorm(0, 0.0001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}
```

All we did to change the model is break apart the distribution specification of `a[j]` into two lines and model `g0, g1, g2` with noninformative normal prior distributions in the usual way.

To run the model, we proceed in the same manner as above. First, this requires creating the group-level variables for treatment and age at baseline. These must be specified at the group level, so we should only have as many of these observations as there are unique children.

```
# Creating the new variables (group-level predictors)

# Treatment (t)
t <- rep(NA, J)
for (i in unique(data.noNA.CD4$newpid)){
t[i] <- unique(data.noNA.CD4$treatmnt[which(data.noNA.CD4$newpid == i)])
}
t <- t[!is.na(t)]

# Baseage (ba)
ba <- rep(NA, J)
for (i in unique(data.noNA.CD4$newpid)){
ba[i] <- unique(data.noNA.CD4$baseage[which(data.noNA.CD4$newpid == i)])
}
ba <- ba[!is.na(ba)]
```

As above, I specify the data, list of starting values for the algorithm, and vector of parameters that will be passed to the `jags()` function. Then, I also specify the model, saving it to the `hiv.mod2` object. Finally, I run JAGS, passing it all the necessary objects.

```
# Lists the data that will be contained in our JAGS mode (now, we also model t and ba)
hiv.data2 <- list("n","J","y","child","x", "t", "ba")

# Function to return list of starting values (now, we also need priors on the g0,g1,g2)
hiv.inits2 <- function(){
  list(a=rnorm(J), b=rnorm(1), g0=rnorm(1), g1=rnorm(1), g2=rnorm(1),
       sigma.y=runif(1), sigma.a=runif(1))}

# This is vector of names of parameters we want to save (we now also want to report g0, g1, g2)
hiv.parameters2 <- c("a", "b", "g0", "g1", "g2", "sigma.y", "sigma.a")

# The model itself
hiv.mod2 <- 'model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- a[child[i]] + b*x[i]
  }
  b ~ 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)
    a.hat[j] <- g0 + g1*t[j] + g2*ba[j]
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dnorm(0, 0.0001)
  g2 ~ dnorm(0, 0.0001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}'

# This is the actual JAGS run
hiv.2 <- jags(data=hiv.data2, inits=hiv.inits2, parameters.to.save=hiv.parameters2,
              model.file=textConnection(hiv.mod2), n.chains=3, n.iter=500, DIC=F)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 1072
   Unobserved stochastic nodes: 256
   Total graph size: 5877

Initializing model
```

To examine the results of our model, we can again turn to examining an abbreviated version of the model's output in Table 2 below. This table takes the same general form as the one above. We see that we have new parameters estimated in this model – γ_0, γ_1, and γ_2, corresponding to the three parameters we modeled our α_j as a function of. We see that, as before, the child-specific intercept estimates vary as expected. We see that, as in the previous model, $\hat{\beta}$ is negative and precisely estimated, suggesting that `rootCD4` levels go down over time. We see that the coefficient estimate on the group-level parameter for the treatment, γ_1, is positive but not statistically distinguishable from zero. This suggests that the presence of the treatment has no effect on the child's intercept. We see that the coefficient estimate on the group-level parameter for baseage, γ_2, is negative and statistically distinguishable from zero. This suggests that older children have lower initial (intercept) levels of `rootCD4`.

Table 2: Group-Level Predictors Model

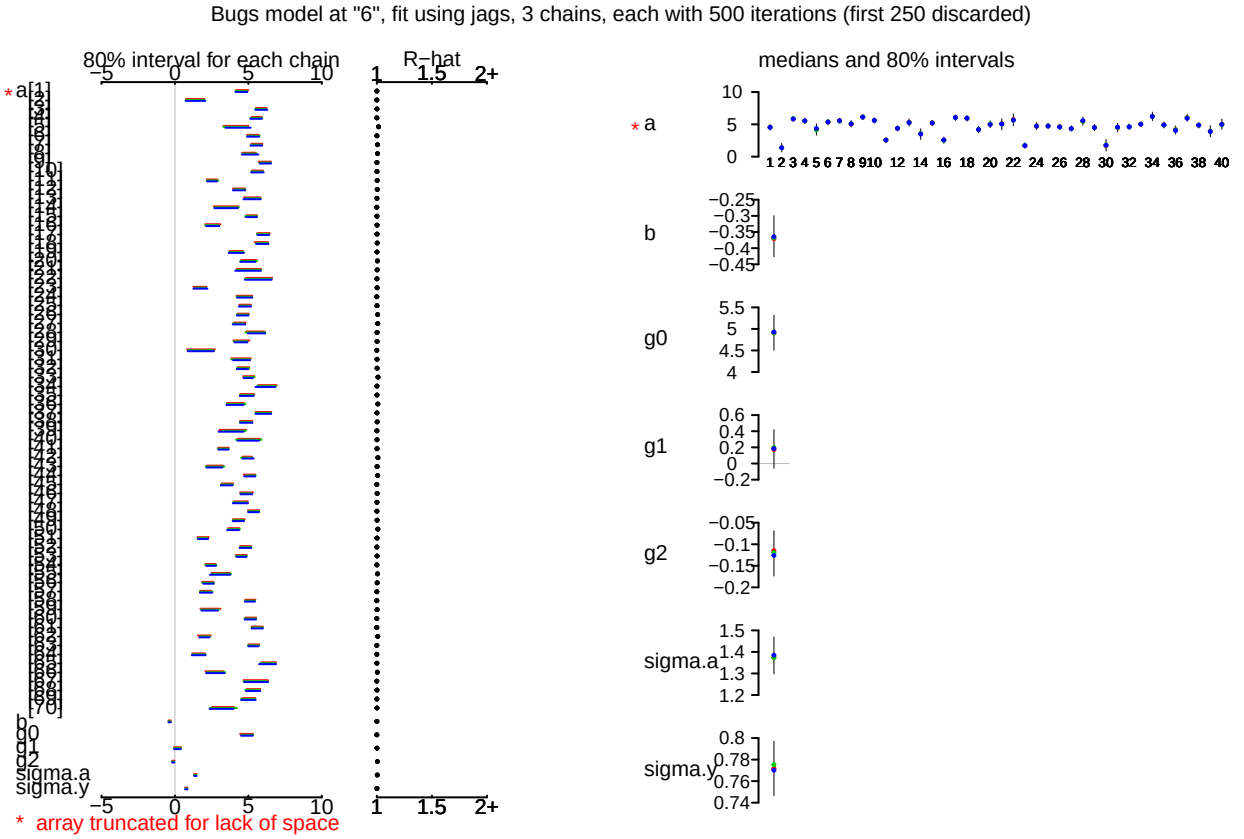
Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
α_1	4.544	0.350	3.852	4.314	4.552	4.769	5.219	1.000	750
α_2	1.373	0.524	0.398	1.014	1.372	1.730	2.418	0.999	750
⋮									
α_{250}	3.902	0.527	2.856	3.553	3.904	4.274	4.909	1.000	750
β	-0.365	0.057	-0.472	-0.404	-0.365	-0.326	-0.251	1.001	750
γ_0	4.906	0.311	4.276	4.712	4.899	5.116	5.538	1.001	750
γ_1	0.182	0.181	-0.179	0.074	0.189	0.305	0.552	1.001	750
γ_2	-0.120	0.038	-0.197	-0.144	-0.121	-0.094	-0.044	1.002	750
σ_α	1.381	0.072	1.243	1.333	1.382	1.427	1.525	1.008	310
σ_y	0.773	0.019	0.735	0.761	0.772	0.786	0.810	1.004	750

DIC info (using the rule, pD = var(deviance)/2): pD = 345.1 and DIC = 2835.9

To compare this model and our previous model, we can look at how the two differ in their estimates of the within-child (σ_y) and across-child (σ_α) standard deviations. The current model estimates the within-child standard deviation to be about 0.773, and the across-child standard deviation to be about 1.381. The previous model's estimates were 0.77 and 1.40, respectively. So, we see that this model, just like the previous one, suggests that there is less variation within a child on their CD4 levels than across children. We also see that our new model, with group-level predictors, is able to explain a little more of the across-child differences in `rootCD4` level than the previous model.

Just as before, we can also use the built-in plotting function to examine the JAGS simulations of our multilevel regression with varying intercepts and two group-level predictors.

```
plot(hiv.2)
```



c.

To understand how the results of these models compare to the fits from `lmer()`, we first need to run the same models using `lmer()`. I do so below.

```
# lmer() fit of only varying intercepts
inddummies <- as.factor(data.noNA.CD4$newpid)

lmer.1 <- lmer(rootCD4 ~ time + (1 | inddummies), data=data.noNA.CD4)
#display(lmer.1, digits=3) # Coefficients very similar, st. dev estimates very similar

# lmer() fit of varying intercepts with group-level predictors
```

```
lmer.2 <- lmer(rootCD4 ~ time + treatmnt + baseage + (1 | inddummies), data=data.noNA.CD4)
#display(lmer.2, digits=3) # Again, very similar coefficients and st. dev estimates
```

Table 3 shows the results from `lmer.1`, the model with varying intercepts and time as an individual-level predictor. I will go step-by-step through the estimates from this model and compare them to the results from the JAGS model reported above in 1. We see that, first off, the coefficient estimate for the time variable in the model is -0.366, with a standard error of 0.054. This is nearly identical to the estimates from our JAGS run, which were -0.366 and 0.056. We see that the estimate for the average intercept in this model is 4.763 with a standard error of 0.096. Our JAGS model has also this estimate and standard error at 4.763 and 0.096 – identical. We see that $\hat{\sigma_{alpha}} = 1.399$ in our `lmer()` model, and $\hat{\sigma_y} = 0.772$. The JAGS model estimated these to be 1.400 and 0.774. So, we see that the results from JAGS and from `lmer()` for the varying-intercept model are very similar, and in some cases identical.

Table 3: lmer Fit of Varying-Intercepts Model

Variable	Estimate	Std. Err.
α	4.763	0.096
β_{time}	-0.366	0.054
Variable		Std. Dev.
σ_α		1.399
σ_y		0.772

number of obs: 1072, groups: inddummies, 250
AIC = 3148.8, DIC = 3126.9, deviance = 3133.9

Table 4 shows the results from `lmer.2`, the model that adds group-level predictors to our varying-intercepts. model. Much as we saw similarities between `lmer.1` and the varying-intercepts model run in JAGS, we see similarities between `lmer.2` and the model with group-level predictors run in JAGS. The estimate for the coefficient on time in this model is -0.362 with a standard error of 0.054, whereas in the model estimated using JAGS it was -0.365 and 0.057, quite similar. The estimates for σ_α and σ_y are likewise nearly identical (to at least two decimal places). We see that overall, JAGS and the `lmer()` function produce similar results for these fairly basic multilevel models.

Table 4: lmer Fit of Varying-Intercepts Model with Group-level Predictors

Variable	Estimate	Std. Err.
α	4.906	0.317
β_{time}	-0.362	0.054
γ_{treatmnt}	0.180	0.183
γ_{baseage}	-0.119	0.040
Variable		Std. Dev.
σ_α		1.375
σ_y		0.773

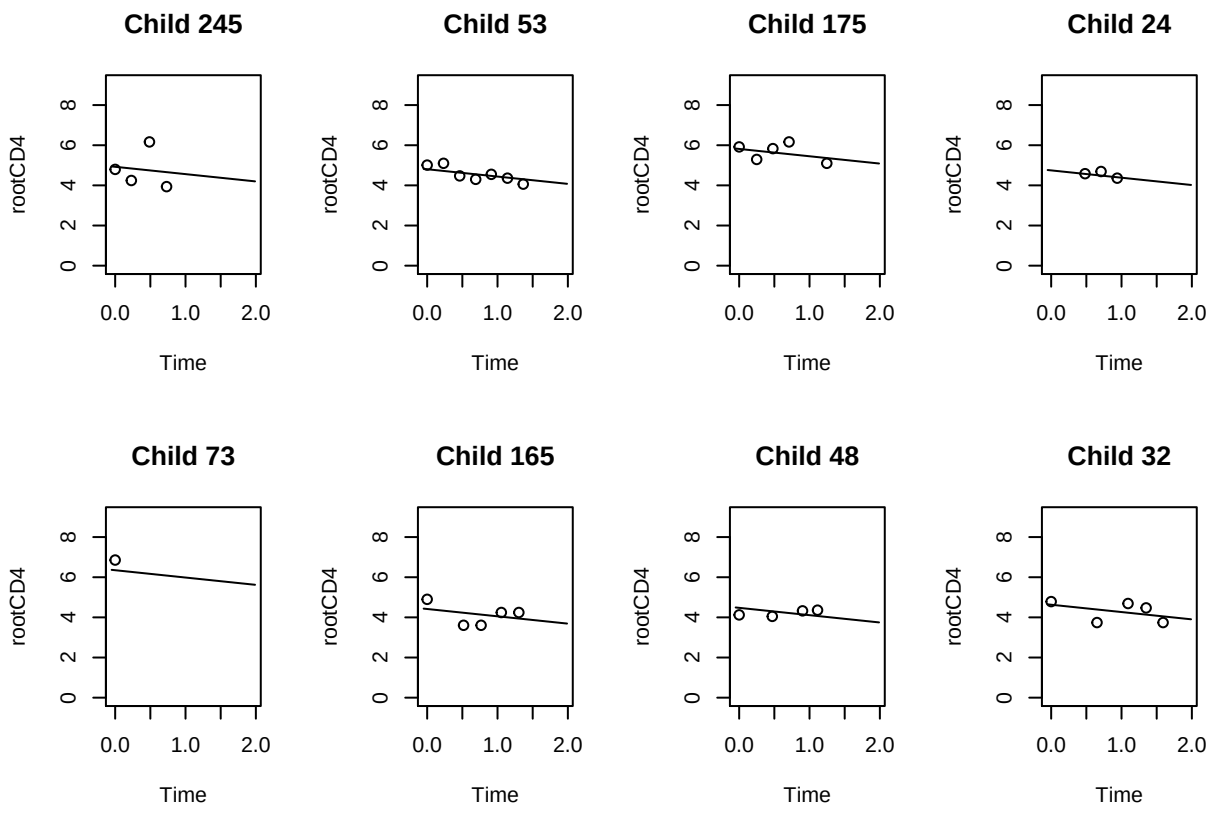
number of obs: 1072, groups: inddummies, 250
AIC = 3149.2, DIC = 3110.9, deviance = 3124.1

d.

Now, I move to summarizing the results of our multilevel models fit using JAGS graphically in a manner similar to Section 16.3 of the textbook. This section has two sets of graphical summaries of models fit using JAGS. The first is the built-in plotting summaries accessed through the `plot()` function. These are presented above in parts (a) and (b). Second, the book suggests plotting, much as we did in Chapter 12 of the book, the estimated regression lines for a sample of 8 children based upon the multilevel model, as well as plotting

the estimated intercept of the child given the amount of observations contained within each child. I plot both of these below. First, I begin by plotting the predicted regression lines for 8 of the children using the predictions from our model. The code to do so is listed below. We see that, generally, the models fit well.

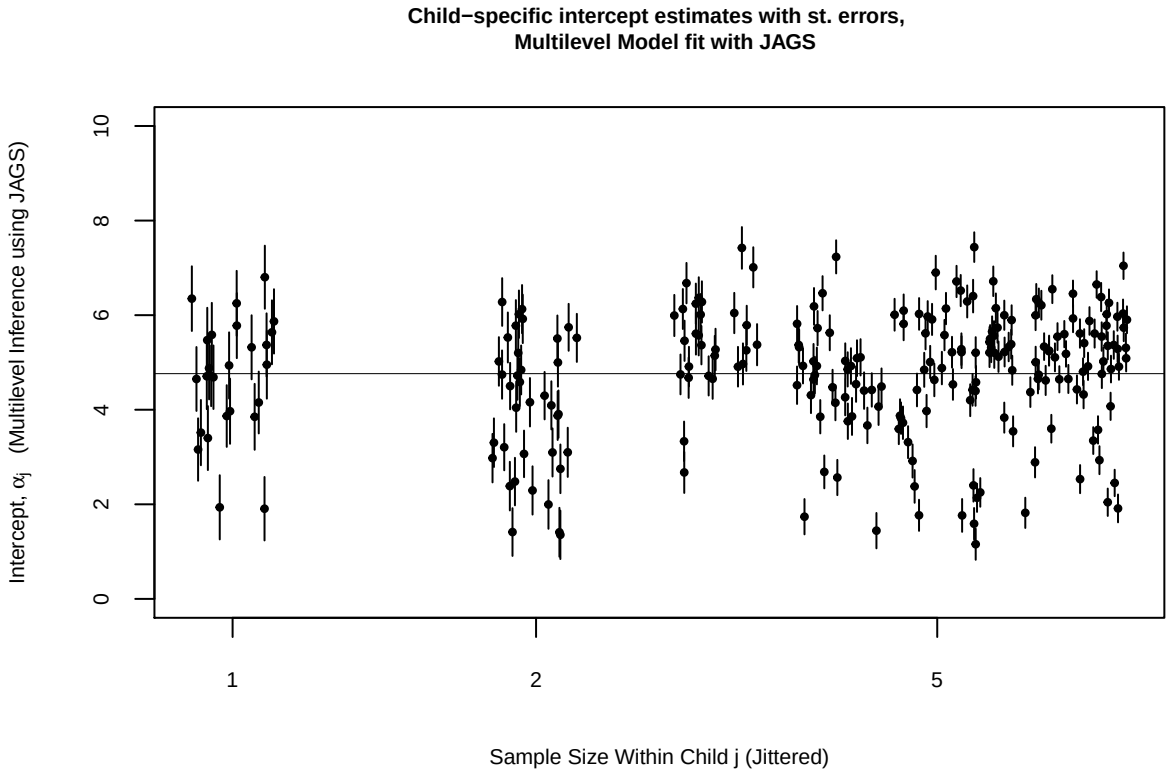
```
# Summarizing multilevel inferences graphically
set.seed(844)
display8 <- c(sample(unique.child, 8)) # Sample 8 children to plot
x.range <- c(min(x) - 0.05, max(x) + 0.05) # Set range of x, y axes
y.range <- c(min(y) - 0.05, max(y) + 0.05)

attach.jags(hiv.1) # Attach the multilevel model run in JAGS (so we can access it)
a.multilevel <- rep(NA, J) # Get median intercept estimate for each child
for (j in 1:J){
  a.multilevel[j] <- median(a[,j])}
b.multilevel <- median(b) # Get median slope estimate for each child

# Plot the lines along with the observation points
par (mfrow=c(2,4))
for (j in display8){
  plot (x[child==j], y[child==j], xlim=x.range, ylim=y.range,
    xlab="Time", ylab="rootCD4", main=paste("Child",unique.child[j]))
  curve (a.multilevel[j] + b.multilevel*x, lwd=1, col="black", add=TRUE)
}
```



Below, I plot the estimated child-level intercepts α_j as a function of the (jittered) sample size of child j, that is, how many observations are contained within child j. Unlike in the book's example, we do not see too

much of a trend in the variability of these estimates across the values of the sample size. Nonetheless, it does appear that the uncertainty around these estimates decreases as the sample size increases. This makes sense – with more observations, we are better able to determine the intercept value of a given child.

```
# Determining how many observations within each child J
sample.size <- rep(NA, J)
for (i in unique(data.noNA.CD4$newpid)){
sample.size[i] <- length(which(data.noNA.CD4$newpid == i))
}
sample.size <- sample.size[!is.na(sample.size)]

# Jittering to help with plotting visualization
sample.size.jittered <- sample.size * exp(runif(J, -.1, .1))

# Plotting
par(mfrow=c(1,1))
plot (sample.size.jittered, a.multilevel, xlab="Sample Size Within Child j (Jittered)",
  ylim=c(0,10), ylab=expression (paste ("Intercept, ", alpha[j],
  "   (Multilevel Inference using JAGS)")), pch=20, log="x", cex=0.7, cex.axis=0.7, cex.main=0.7,
  cex.lab=0.7, main="Child-specific intercept estimates with st. errors,
  Multilevel Model fit with JAGS")
for (j in 1:J){
  lines (rep(sample.size.jittered[j],2), median(a[,j]) + c(-1,1)*sd(a[,j]))}
abline (4.763, 0, lwd=.5)
```



16.8

a.

To begin, as a reference, I fit the "regular" version of the regression of the JAGS model of radon data as a function of floor, with varying intercepts that depend on uranium as the county-level predictor. By "regular," I simply mean using the usual priors recommended by Gelman & Hill.

```
# Regular version
# Setting up the data
srrs2 <- read.table("srrs2.dat", header=TRUE, sep=",")
mn <- srrs2$state == "MN"
radon <- srrs2$activity[mn]
y <- log(ifelse(radon==0, 0.1, radon))
n <- length(radon)
x <- srrs2$floor[mn]

# Creating county indicators
srrs2.fips <- srrs2$stfips*1000 + srrs2$cntyfips
county.name <- as.vector(srrs2$county[mn])
uniq.name <- unique(county.name)
J <- length(uniq.name)
county <- rep(NA, J)
for (i in 1:J){
  county[county.name == uniq.name[i]] <- i
}

# Creating county-level variable for uranium
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)

# The model itself
rad.1 <- '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, .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)
    a.hat[j] <- g0 + g1*u[j]
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dnorm(0, 0.0001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}'
```

```
# Lists the data that are written to a file to be read by JAGS
rad.data <- list("n","J","y","county","x", "u")

# Function to return list of starting values of algorithm
rad.inits <- function(){
  list(a=rnorm(J), b=rnorm(1), g0=rnorm(1), g1=rnorm(1),
       sigma.y=runif(1), sigma.a=runif(1))}

# This is vector of names of parameters we want to save from the JAGS run
rad.parameters <- c("a", "b", "g0", "g1", "sigma.y", "sigma.a")
```

Finally, I run the model using the `jags()` function, passing it the necessary arguments. I choose to run 1000 iterations of the model because the $\hat{R}$ values for some of the parameters were over 1.1 after only 500 iterations. The results from this model are presented below in Table 5. We see that our estimates of the parameters are all statistically distinguishable from zero. As expected, the coefficient on floor, β is negative – as we move from the basement to the first floor, we expect the radon count to decrease. Also as expected, there is variability across counties in the level of radon measured in houses. We see that the across-group variability is substantial, but is much less than the within-group variability. We see that the $\hat{R}$ values are all less than 1.1, suggesting our simulations converged well.

Table 5: Radon Model, Standard Priors

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
α_1	0.937	0.157	0.611	0.837	0.940	1.033	1.252	1.027	160
α_2	0.863	0.089	0.694	0.802	0.861	0.925	1.035	1.001	1500
⋮									
α_{85}	-0.672	0.073	-0.811	-0.719	-0.673	-0.624	-0.526	1.001	1500
β	-0.669	0.071	-0.806	-0.715	-0.674	-0.622	-0.525	1.001	1500
γ_0	1.466	0.040	1.388	1.439	1.467	1.494	1.546	1.019	180
γ_1	0.723	0.086	0.542	0.671	0.725	0.776	0.888	1.007	310
σ_α	0.153	0.045	0.075	0.122	0.150	0.184	0.243	1.056	86
σ_y	0.760	0.018	0.725	0.747	0.760	0.772	0.797	1.005	380

```
# This is the actual JAGS run
#mod.1 <- jags(data=rad.data, inits=rad.inits, parameters.to.save=rad.parameters,
#              model.file=textConnection(rad.1), n.chains=3, n.iter=1000, DIC=F)
```

Now, I assign normal prior distributions with mean 5 and standard deviation 1000 to the coefficients for floor and uranium. This is a departure from the usual practice of assigning normal prior distributions with mean 0 and standard deviation 100 to these coefficients. We are making the standard deviation larger, which makes our distribution even less informative, but we do move the mean away from 0. The only alterations to the model require us to write `b` $\sim$ `dnorm (5, 0.000001)` and `g1` $\sim$ `dnorm(5, 0.000001)`.

We can take a look at the estimates of this model below in Table 6. As we are changing the prior distributions on β and $\gamma 1$, I will focus only on comparing those coefficient estimates (and the uncertainty around them) to the usual model. We see that the estimate of β from our new model is -0.669, with a standard error of 0.068. This is very similar to that of the previous model, which had an identical point estimate of -0.669 and a slightly higher standard error of 0.071. We see that the estimate of γ_1 from our new model is 0.712, with a standard error of 0.097. This, again, is similar to that of the previous model, which had a point estimate of 0.723 and standard error of 0.086. So, this suggests that even though we moved the mean up from zero in our noninformative prior, the larger standard deviation of the prior distribution allowed the algorithm to converge to roughly the same parameter values as the usual priors, with generally similar estimation uncertainty.

Table 6: Radon Model, Normal Priors with mean=5, st. dev=1000

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
β	-0.669	0.068	-0.801	-0.714	-0.670	-0.621	-0.537	1.005	430
γ_1	0.712	0.097	0.520	0.649	0.713	0.775	0.906	1.013	200

```
# Assigning normal prior distributions with mean 5 and standard deviation 1000
# to coefficients for floor and uranium
# b ~ dnorm(5, 0.000001); g1~ dnorm(5, 0.000001)
rad.2 <- '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, 0.000001)
  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] <- g0 + g1*u[j]
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dnorm(5, 0.000001)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}'

# JAGS run
#mod.2 <- jags(data=rad.data, inits=rad.inits, parameters.to.save=rad.parameters,
#                model.file=textConnection(rad.2), n.chains=3, n.iter=1000, DIC=F)
```

b.

Now, I assign normal prior distributions to to the coefficients for floor and uranium with a mean of 0 and a standard deviation of 0.1. This is a fairly limiting prior distribution, as we are telling the algorithm that we expect the coefficient to, roughly, range from (-0.1, 0.1). As we know that our prior models estimated these coefficients to be about -0.7 and 0.7, we expect that this prior will pull our estimates toward zero. The alterations to the model require us to write `b` $\sim$ `dnorm(0,100)` and `g1` $\sim$ `dnorm(0,100)`.

We examine the changes to the coefficient estimates and their standard errors, show in Table 7 below. We see that the coefficient estimates are pulled toward zero in comparison to our previous model. The estimated coefficient on β is -0.454, with a standard error of 0.058. This is in comparision to the standard model, which gave us an of -0.669 and a standard error of 0.071. So, we see that it appears that the prior restricts the coefficient to be closer to zero than a true noninformative prior would allow. Similarly, the estimated coefficient on γ_1 is 0.336, with a standard error of 0.082, whereas in the standard model it was 0.723 with a standard error of 0.086. We see again that the prior restricts the coefficient to be closer to zero than in our standard model.

```
# Now, assigning b ~ dnorm(0,100); g1~ dnorm(0,100)
# This should restrict how far estimate can travel from 0
```

Table 7: Radon Model, Normal Priors with mean=0, st. dev=0.1

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
β	-0.454	0.058	-0.564	-0.494	-0.454	-0.414	-0.343	1.002	1500
γ_1	0.336	0.082	0.165	0.284	0.339	0.393	0.489	1.011	290

```
rad.3 <- '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] <- g0 + g1*u[j]
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dnorm(0, 100)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}'

# JAGS run
#mod.3 <- jags(data=rad.data, inits=rad.inits, parameters.to.save=rad.parameters,
#                model.file=textConnection(rad.3), n.chains=3, n.iter=1000, DIC=F)
```

c.

Now, I move to trying normal prior distributions with mean 5 and standard deviation 1. This is a much wider standard deviation than the 0.1 in part (b), but we do adjust the mean up to 5. The code to implement this is `b ~ dnorm(5,1)` and `g1 ~ dnorm(5,1)`.

The results of this model are presented in Table 8. We see that the results from this model are fairly similar to those from the standard model. The estimated coefficient on β is -0.645, with a standard error of 0.068; the estimated coefficient on γ_1 is 0.759 with a standard error of 0.095. These are fairly similar to our coefficient estimates from the standard model (-0.669 and 0.723), although the similarity is only to the first decimal point. So, it appears that this prior is not far enough off as to cause the model to reach simulation values that are far from those obtained using the standard model.

Table 8: Radon Model, Normal Priors with mean=5, st. dev=1

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
β	-0.645	0.068	-0.779	-0.691	-0.645	-0.598	-0.511	1.005	520
γ_1	0.759	0.095	0.572	0.693	0.755	0.825	0.950	1.004	640

```
# Now, trying b ~ dnorm(5,1); g1~ dnorm(5,1)

rad.4 <- '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] <- g0 + g1*u[j]
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dnorm(5, 1)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}'

# JAGS run
# mod.4 <- jags(data=rad.data, inits=rad.inits, parameters.to.save=rad.parameters,
#                 model.file=textConnection(rad.4), n.chains=3, n.iter=1000, DIC=F)
```

d.

Now, I move to modeling the prior distributions of the coefficients on floor and uranium as t-disributed with mean 5, standard deviation 1, and 4 degrees of freedom. We know that the t-distribution is somewhat normal-looking but with heavier tails. So, we may expect that these results are fairly similar to those in part (c), where we had normal priors with mean 5 and standard deviation 1. The code to implement this change is `b ~ dt(5, 1, 4)` and `g1 ~ dt(5,1,4)`.

The results from this model are shown below in Table 9. I use 1500 simulations as using only 1000 fails to provide adequate effective n values. We see that the model's estimates yet again fairly closely resemble those of the standard model. The estimated coefficient on β is -0.667, with a standard error of 0.071; the estimated coefficient on γ_1 is 0.733 with a standard error of 0.094. These are very similar to our coefficient estimates from the standard model (-0.669 and 0.723), accurate to the second and first decimal point, respectively.

Table 9: Radon Model, t-distributed Priors with mean=5, st. dev=1, df=4

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
β	-0.667	0.071	-0.801	-0.715	-0.668	-0.618	-0.524	1.018	120
γ_1	0.733	0.094	0.542	0.670	0.737	0.797	0.916	1.005	470

```
# Now, trying b ~ dt(5, 1, 4) and g1~dt(5,1,4)

rad.5 <- '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] <- g0 + g1*u[j]
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dt(5,1,4)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}'

# JAGS run
#mod.5 <- jags(data=rad.data, inits=rad.inits, parameters.to.save=rad.parameters,
#                model.file=textConnection(rad.5), n.chains=3, n.iter=1500, DIC=F)
```

e.

Finally, I move to specifying Uniform(-100,100) and Uniform(-1,1) prior distributions. We again change the distribution type of our model. The code to do so is `b ~ dunif(-100, 100)` and `g1 ~ dunif(-100, 100)` for the Uniform(-100,100) distribution, and `b ~ dunif(-1, 1)` and `g1 ~ dunif(-1, 1)` for the Uniform(-1,1) distribution. I run these models below. For the Uniform(-1,1) priors, the model fails to run without changing the initial values for β and γ_1 – as they are drawn from the standard normal, they don't quite fit in line with the Uniform(-1,1) prior. Therefore, I change the initial values to come from the uniform distribution.

The results from these models are shown in Tables 10 and 11. We see that yet again, these priors did little to stop the algorithm from converging on generally the same estimates of the parameters as they did in the standard model. The β estimates are -0.669 and -0.673, whereas in the standard model it was -0.669. The γ_1 estimates are 0.725 and 0.729, whereas in the standard model it was 0.723.

Table 10: Radon Model, Uniform Priors over -100 to 100

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
β	-0.669	0.070	-0.808	-0.717	-0.669	-0.622	-0.535	1.003	780
γ_1	0.725	0.084	0.557	0.672	0.727	0.780	0.886	1.002	890

Table 11: Radon Model, Uniform Priors over -1 to 1

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
β	-0.673	0.070	-0.803	-0.720	-0.675	-0.627	-0.537	1.006	340
γ_1	0.729	0.101	0.536	0.659	0.736	0.799	0.923	1.012	160

```
# Now trying b ~ dunif(-100, 100) and g1 ~ dunif(-100, 100)

rad.6 <- '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] <- g0 + g1*u[j]
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dunif(-100, 100)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}'

# JAGS run
#mod.6 <- jags(data=rad.data, inits=rad.inits, parameters.to.save=rad.parameters,
#                model.file=textConnection(rad.6), n.chains=3, n.iter=1000, DIC=F)

# Now trying b ~ dunif(-1, 1) and g1 ~ dunif(-1, 1)

rad.7 <- '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] <- g0 + g1*u[j]
  }
  g0 ~ dnorm(0, 0.0001)
  g1 ~ dunif(-1, 1)
  tau.a <- pow(sigma.a, -2)
  sigma.a ~ dunif (0, 100)
}'

# JAGS run
# Won't run initally -- jags realizes distribution not consistent with initial values
# So, set new initial value
rad.inits <- function(){
  list(a=rnorm(J), b=runif(1), g0=rnorm(1), g1=runif(1),
       sigma.y=runif(1), sigma.a=runif(1))}
#mod.7 <- jags(data=rad.data, inits=rad.inits, parameters.to.save=rad.parameters,
#                model.file=textConnection(rad.7), n.chains=3, n.iter=1000, DIC=F)
```

f.

So, what did we learn from all of these different specifications of prior distributions? First, it appears that changing both the mean/standard deviations or the form of the prior distribution itself does not typically change the final estimates the algorithm converges upon. This suggests that, as long as our priors do not specify wholly inaccurate mean values, or have very restricting standard deviations, our model is unlikely to have too much trouble in converging on a good estimate of the coefficient of interest. Indeed, the only time our model had much trouble was when the standard deviation was restricted to be incredibly small – only 0.01. Even then, our coefficient estimate was able to come reasonably close to that of the "standard" noninformative

model, but it was pulled towards zero due to the restrictive prior distribution. So, the lesson is to try to pick fairly noninformative priors, and we know that as long as it is fairly reasonable and noninformative, our we will likely settle on a reasonable result.

17.1

In this problem, I add to the model from exercise 16.3(b), now allowing for varying slopes on the time variable as well as varying intercepts. We recall that this model is a model of `rootCD4`, the square root of a child's square root CD4 percentage at a given hospital visit. We now allow the intercepts and slope on time to vary by child. We have group-level predictors baseage and treatment for both varying coefficients. We can write this varying-intercepts, varying-slopes model in JAGS with the following code:

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

  for(j in 1:J){
    a[j] <- B[j,1]
    b[j] <- B[j,2]
    B[j, 1:2] ~ dmnorm(B.hat[j,], Tau.B[,])
    B.hat[j,1] <- gamma.0.a + gamma.1.a*treatmnt[j] + gamma.2.a*baseage[j]
    B.hat[j,2] <- gamma.0.b + gamma.1.b*treatmnt[j] + gamma.2.b*baseage[j]
  }
  gamma.0.a ~ dnorm(0, .0001)
  gamma.1.a ~ dnorm(0, .0001)
  gamma.2.a ~ dnorm(0, .0001)
  gamma.0.b ~ dnorm(0, .0001)
  gamma.1.b ~ dnorm(0, .0001)
  gamma.2.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)
}
```

We allow there to be varying intercepts and varying slopes on the `time` predictor by assigning different `a` and `b` parameters for each child j, saved to the `B` matrix. We introduce the group-level predictors in the model by defining `B.hat[j,1]` and `B.hat[j,2]` accordingly. The code beginning with `Tau.B[1:2,1:2]` allows us to model the multilevel correlation between intercepts and slopes.

```
# Loading HIV data
hiv <- read.csv("allvar.csv", header=TRUE)
# Square root transformation of the CD4PCT
```

```
hiv$rootCD4 <- sqrt(hiv$CD4PCT)
# Creation of time variable
hiv$time <- hiv$visage - hiv$baseage

# Removing those cases that have NAs for the DV or the time variable
data.noNA.CD4 <- hiv[complete.cases(hiv[,4]),]
data.noNA.CD4 <- data.noNA.CD4[complete.cases(data.noNA.CD4[,11]),]

# Creating n
n <- length(data.noNA.CD4$rootCD4)

# Creating J
unique.child <- unique(data.noNA.CD4$newpid)
J <- length(unique.child)

# Creating y
y <- data.noNA.CD4$rootCD4

# Creating time variable
time <- data.noNA.CD4$time

# Creating child indicator
child <- rep(NA, J)
for (i in 1:J){
  child[data.noNA.CD4$newpid == unique.child[i]] <- i
}

# Creating the group-level predictors
# treatmnt
treatmnt <- rep(NA, J)
for (i in unique(data.noNA.CD4$newpid)){
treatmnt[i] <- unique(data.noNA.CD4$treatmnt[which(data.noNA.CD4$newpid == i)])
}
t <- treatmnt[!is.na(treatmnt)]

# baseage
baseage <- rep(NA, J)
for (i in unique(data.noNA.CD4$newpid)){
baseage[i] <- unique(data.noNA.CD4$baseage[which(data.noNA.CD4$newpid == i)])
}
ba <- baseage[!is.na(baseage)]
```

```
# Lists the data that will be contained in our JAGS model
hiv.data <- list("n","J","y","child","time", "t", "ba")

# Function to return list of starting values of algorithm
hiv.inits <- function(){list(B=array(rnorm(2*J), c(J,2)), g0a=rnorm(1), g1a=rnorm(1),
  g2a=rnorm(1), g0b=rnorm(1), g1b=rnorm(1), g2b=rnorm(1),
  sigma.y=runif(1), sigma.a=runif(1))}

# Vector of names of parameters we want to save from the JAGS run
hiv.parameters <- c("a", "b", "sigma.y", "sigma.a", "rho", "g0a", "g1a", "g2a", "g0b",
                    "g1b", "g2b", "sigma.b")
```

```
# The model
hiv.mod <- 'model{
  for(i in 1:n){
    y[i] ~ dnorm(y.hat[i], tau.y)
    y.hat[i] <- a[child[i]] + b[child[i]]*time[i]
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif(0, 100)

  for(j in 1:J){
    a[j] <- B[j,1]
    b[j] <- B[j,2]
    B[j, 1:2] ~ dmnorm(B.hat[j,], Tau.B[,])
    B.hat[j,1] <- g0a + g1a*t[j] + g2a*ba[j]
    B.hat[j,2] <- g0b + g1b*t[j] + g2b*ba[j]
  }
  g0a ~ dnorm(0, .0001)
  g1a ~ dnorm(0, .0001)
  g2a ~ dnorm(0, .0001)
  g0b ~ dnorm(0, .0001)
  g1b ~ dnorm(0, .0001)
  g2b ~ 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)
}'

# This is the actual JAGS run
set.seed(1)
hiv <- jags(data=hiv.data, inits=hiv.inits, parameters.to.save=hiv.parameters,
                model.file=textConnection(hiv.mod), n.chains=3, n.iter=1500, DIC=F)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 1072
   Unobserved stochastic nodes: 260
   Total graph size: 7869

Initializing model
```

I present an abbreviated version of the model's results in Table 12 below. In interpreting the model, we see that the estimates on the child-specific intercepts are differnt – this is to be expected, as we expect that each child will have a different starting point for their `rootCD4` value. We see the same occurs for the child-specific slopes – this also to be expected, as we expect each child has a different time trend in their `rootCD4` level. The estimates of the average intercept and average slope are precisely estimated. We see that the estimated coefficient on $\gamma_{2\alpha}$ is statistically distinguishable from zero, suggesting that baseage significantly influences an

individual child's starting `rootCD4` value. Neither of the coefficients on the group-level predictor for treatment are statistically distinguishable from zero, suggesting that the treatment does not significantly affect the child's starting `rootCD4` value or time trend. The estiamte for ρ is not statistically distinguishable from zero, suggesting that we cannot say distinguish a correlation between the varying intercepts and slopes. Looking at the $\hat{R}$ column, we have evidence that that the algorithm converged reasonably well – none of these values are very close to 1.1. Looking at the n_{eff} column suggests that, generally, we have enough effective simulations to achieve precise estimates of our parameters.

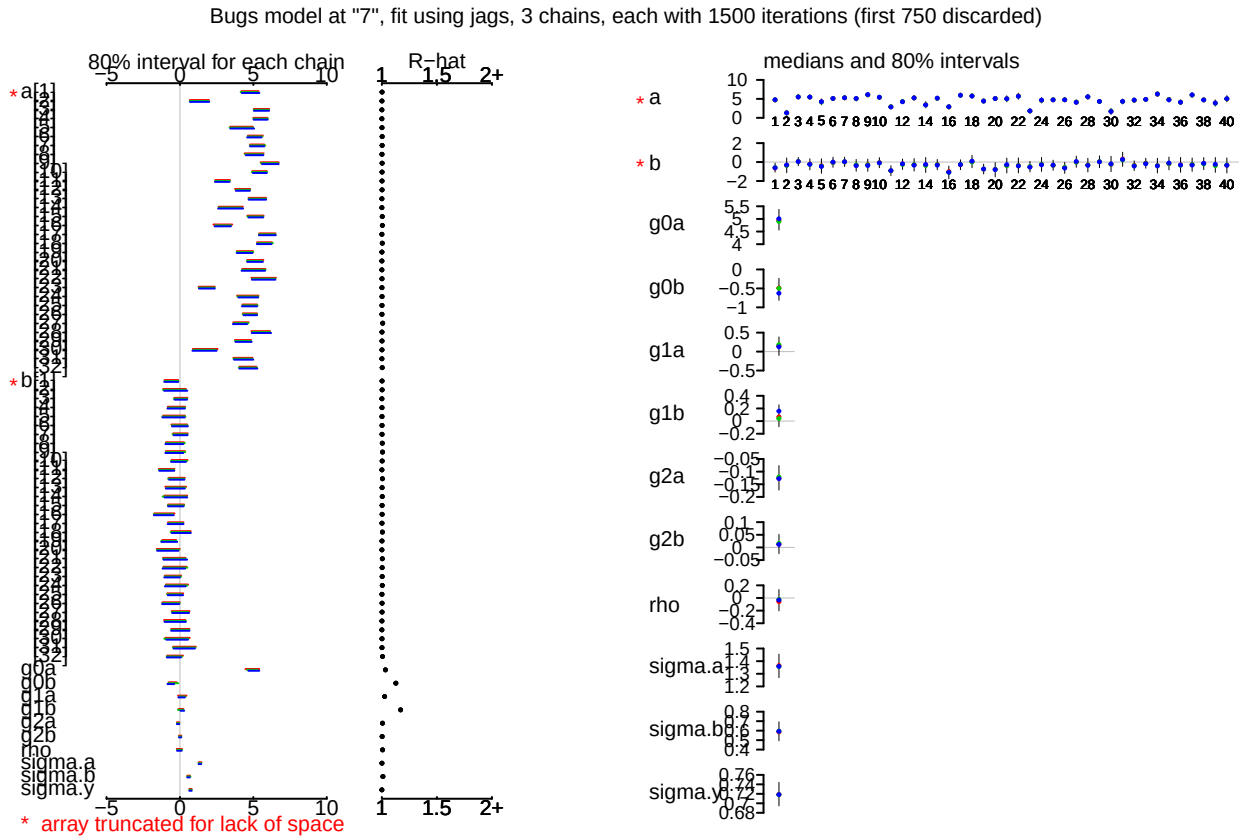
Table 12: Varying Intercepts and Slopes Model

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
α_1	4.760	0.454	3.889	4.449	4.760	5.072	5.637	1.001	2200
α_2	1.283	0.479	0.338	0.966	1.280	1.606	2.221	1.001	2200
⋮									
α_{250}	3.857	0.509	2.820	3.520	3.857	4.189	4.854	1.001	2200
β_1	-0.597	0.358	-1.284	-0.843	-0.604	-0.347	0.092	1.001	2200
β_2	-0.328	0.622	-1.530	-0.751	-0.336	0.089	0.946	1.002	1100
⋮									
β_{250}	-0.268	0.575	-1.356	-0.667	-0.279	0.101	0.930	1.001	2200
σ_α	1.369	0.072	1.232	1.320	1.368	1.417	1.512	1.001	2200
σ_β	0.582	0.079	0.428	0.526	0.582	0.635	0.739	1.003	870
σ_y	0.720	0.020	0.684	0.706	0.719	0.733	0.758	1.001	2200
$\gamma_{0\alpha}$	5.040	0.321	4.380	4.823	5.043	5.263	5.661	1.022	140
$\gamma_{1\alpha}$	0.119	0.179	-0.222	-0.002	0.115	0.233	0.484	1.021	140
$\gamma_{2\alpha}$	-0.133	0.042	-0.214	-0.162	-0.133	-0.103	-0.051	1.008	340
$\gamma_{0\beta}$	-0.552	0.235	-1.035	-0.698	-0.552	-0.400	-0.086	1.045	49
$\gamma_{1\beta}$	0.096	0.132	-0.165	0.007	0.093	0.184	0.352	1.070	35
$\gamma_{2\beta}$	0.016	0.033	-0.046	-0.007	0.016	0.039	0.081	1.003	2200
ρ	-0.040	0.134	-0.306	-0.127	-0.042	0.048	0.233	1.002	1800

DIC info (using the rule, pD = var(deviance)/2): pD = 702.4 and DIC = 3038.2

A second method we can use to aid in interpretation of model results is to examine the built-in graphical summary of a JAGS simulation, accessed through the `plot` function. I plot this summary below. We see that the plot provides us similar results to our tabular analysis above. The left-hand side provides estimates of the (first 30 or so) child-specific intercepts, with 80% uncertainty intervals, for each of the 3 chains. The same is done for the (first 30 or so) child-specific slopes, and for the other parameters estimate in the model. These show us that, generally, we are able to distinguish the intercepts predictions from zero but not the slope predictions. We also see the $\hat{R}$ values plotted for each of these parameter estimates. On the right-hand side, the median estimates of the parameters in the model are plotted with 80% uncertainty bars.

```
plot(hiv)
```



c.

To understand how the results of this model compares to the fit from `lmer()`, we first need to run the same model using `lmer()`. I do so in the chunk of code below. First, this requires creating an indicator variable for each unique child. I fit the model with varying intercepts and slopes with the group-level predictors `lmer.1`. To analyze the results from these fits, and to compare them to our models run using JAGS, we turn to the tables below.

```
# lmer() fit of varying intercept, varying slope with group-level predictors
inddummies <- as.factor(data.noNA.CD4$newpid)
lmer.1 <- lmer(rootCD4 ~ time + treatmnt + baseage + time*baseage + time*treatmnt +
                 (1 + time | inddummies), data=data.noNA.CD4)
```

Table 13 shows abbreviated results from `lmer.1`. I will go step-by-step through the estimates from this model and compare them to the similar model fit in JAGS. We see the estimate for the average intercept $\alpha = 5.008$, is very similar to that of our JAGS model, which was 5.040. We see the estimate for the average slope parameter $\beta = -0.538$, is also similar to that from the JAGS model, which was -0.552. The standard errors of these estimates from our `lmer` model, 0.323 and 0.241, respectively, are also similar than those from JAGS, which are 0.321 and 0.235. We see that $\hat{\sigma_\alpha} = 1.359$ in our `lmer()` model, $\hat{\sigma_y} = 0.718$, and $\hat{\sigma_{\text{time}}} = 0.584$. The JAGS model estimated these to be 1.369, 0.720, and 0.582, respectively – again, very similar estimates. We see that the estimate of ρ from our `lmer` model is -0.004, whereas in JAGS it was -0.40. So, we see that the results from JAGS and from `lmer()` for the varying-intercept model are generally very similar.

Table 13: lmer Fit of Varying Intercepts and Slopes Model

Variable	Estimate	Std. Err.
α	5.008	0.323
β_{time}	-0.538	0.241
γ_{treatmnt}	0.131	0.187
γ_{baseage}	-0.129	0.041
Variable		Std. Dev.
σ_α		1.359
σ_{time}		0.584
σ_y		0.718
ρ		-0.004

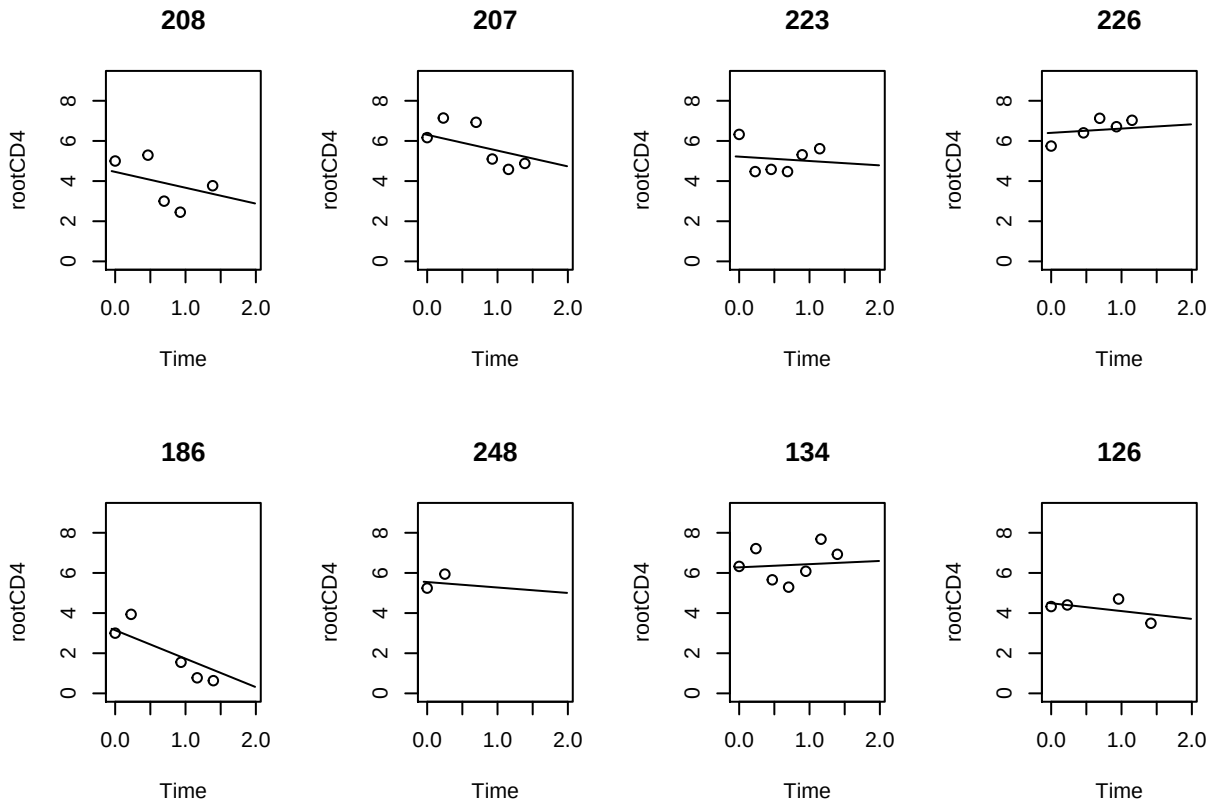
number of obs: 1072, groups: inddummies, 250
AIC = 3133.6, DIC = 3073.6, deviance = 3093.6

d.

Now, I move to summarizing the results of our multilevel model fit using JAGS graphically in a manner similar to Section 16.3 of the textbook. This section has two sets of graphical summaries of models fit using JAGS. The first is the built-in plotting summaries accessed through the `plot()` function. This was presented and discussed above.

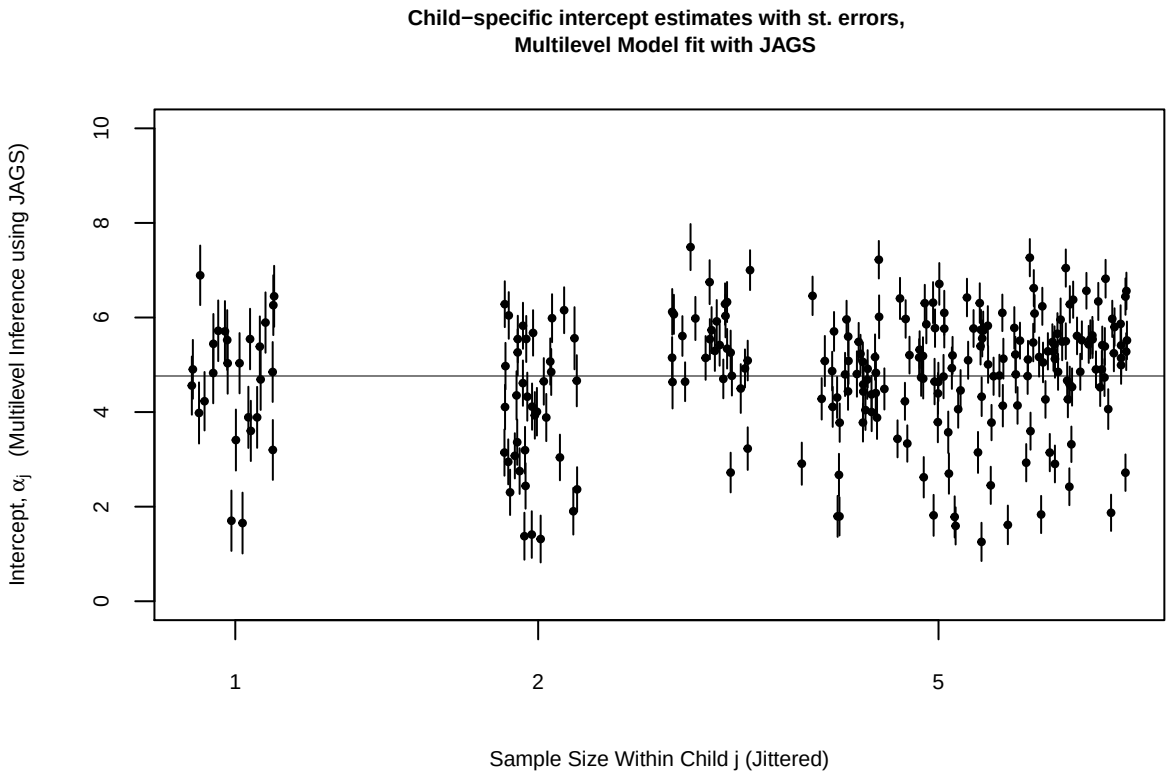
I also plot the estimated regression lines for a sample of 8 children based upon the multilevel model, as well as examining the estimated intercept and slope of the child given the amount of observations contained within each child. I plot both of these below. We see that, generally, the fit of the models for each child is pretty good.

```
# Summarizing multilevel inferences graphically
display8 <- c(sample(unique.child, 8)) # Sample 8 children to plot
x.range <- c(min(time) - 0.05, max(time) + 0.05) # Set range of x, y axes
y.range <- c(min(y) - 0.05, max(y) + 0.05)

attach.jags(hiv) # Attach the multilevel model run in JAGS (so we can access it)
a.multilevel <- b.multilevel <- rep(NA, J) # Get median intercept, slope estimate for each child
for (j in 1:J){
  a.multilevel[j] <- median(a[,j])
  b.multilevel[j] <- median(b[,j])
  }

# Plot the lines along with the observation points
par (mfrow=c(2,4))
for (j in display8){
  plot (time[child==j], y[child==j], xlim=x.range, ylim=y.range,
    xlab="Time", ylab="rootCD4", main=unique.child[j])
  curve (a.multilevel[j] + b.multilevel[j]*x, lwd=1, col="black", add=TRUE)
}
```



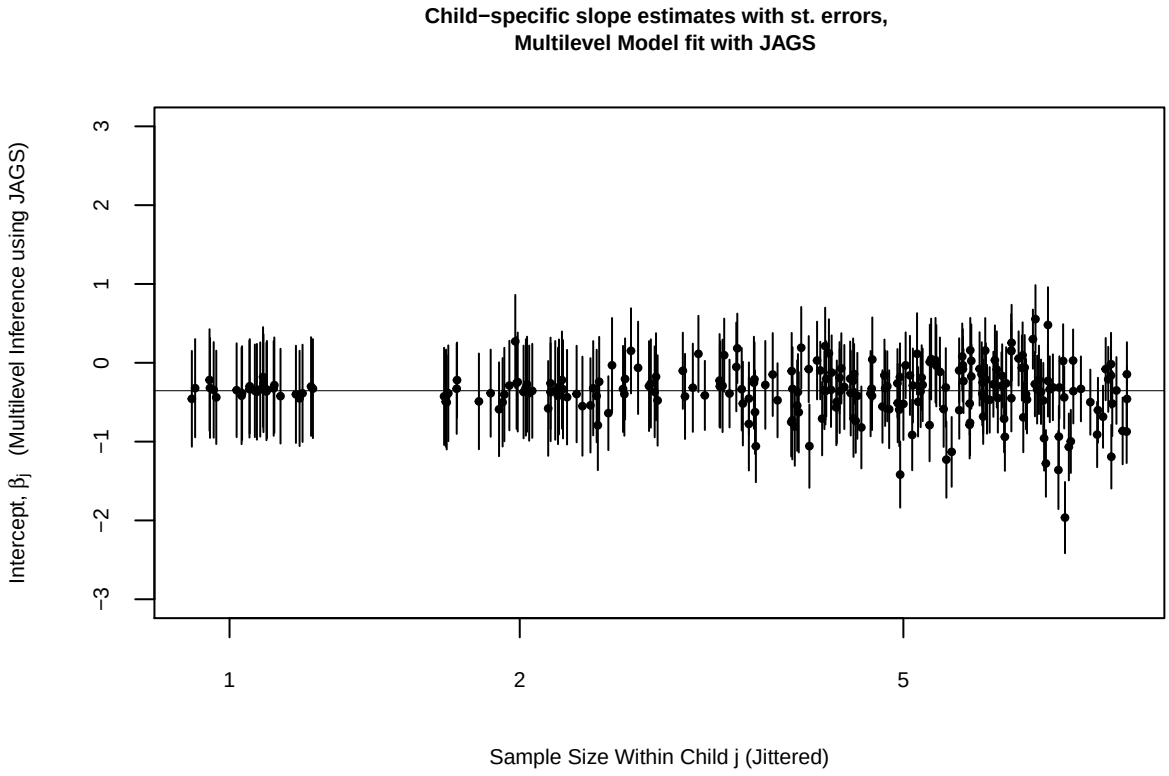
As in part 13.6 of the book, I plot the estimated child-level intercepts α_j as a function of the (jittered) sample size of child j, that is, how many observations are contained within child j. It does appear that the uncertainty around these estimates decreases as the sample size increases. This makes sense – with more observations, we are better able to determine the intercept value of the given child.

```
# Determining how many observations within each child J
sample.size <- rep(NA, J)
for (i in unique(data.noNA.CD4$newpid)){
sample.size[i] <- length(which(data.noNA.CD4$newpid == i))
}
sample.size <- sample.size[!is.na(sample.size)]

# Jittering to help with plotting visualization
sample.size.jittered <- sample.size * exp(runif(J, -.1, .1))

# Plotting
par(mfrow=c(1,1))
plot (sample.size.jittered, a.multilevel, xlab="Sample Size Within Child j (Jittered)",
  ylim=c(0,10), ylab=expression (paste ("Intercept, ", alpha[j],
  "   (Multilevel Inference using JAGS)")), pch=20, log="x", cex=0.7, cex.axis=0.7, cex.main=0.7,
  cex.lab=0.7, main="Child-specific intercept estimates with st. errors,
  Multilevel Model fit with JAGS")
for (j in 1:J){
  lines (rep(sample.size.jittered[j],2), median(a[,j]) + c(-1,1)*sd(a[,j]))}
abline (4.763, 0, lwd=.5)
```



I also repeat the same exercise for the estimated child-level slopse β_j as a function of the (jittered) sample size of child j. Much as above, it appears that the uncertainty around these estimates decreases as the sample size increases, although the effect is not as pronounced as it was for the intercepts. This is likely due to the greater (relative) predictive uncertainty around these estimates, as evident in the table above.

```
sample.size.jittered <- sample.size * exp(runif(J, -.2, .2))
# Plotting
par(mfrow=c(1,1))
plot (sample.size.jittered, b.multilevel, xlab="Sample Size Within Child j (Jittered)",
  ylim=c(-3,3), ylab=expression (paste ("Intercept, ", beta[j],
  "   (Multilevel Inference using JAGS)")), pch=20, log="x", cex=0.7, cex.axis=0.7, cex.main=0.7,
  cex.lab=0.7, main="Child-specific slope estimates with st. errors,
  Multilevel Model fit with JAGS")
for (j in 1:J){
  lines (rep(sample.size.jittered[j],2), median(b[,j]) + c(-1,1)*sd(b[,j]))}
abline (mean(b), 0, lwd=.5)
```



17.5

17.5 I use JAGS to fit the model I set up for the age-guessing data in Exercise 13.4. This exercise had us specify a separate error variance by group, but we were unable to run this model using `lmer`. We can, however, specify this separate error variance using JAGS.

```
model{
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y[group[i]])
    y.hat[i] <- mu + gamma[group[i]] + delta[individual[i]]
  }
  mu ~ dnorm(0, .0001)

  for (k in 1:K){
    delta[k] ~ dnorm(0, tau.delta)
    }
  tau.delta <- pow(sigma.delta, -2)
  sigma.delta ~ dunif(0, 100)

  for(j in 1:J){
    gamma[j] ~ dnorm(0, tau.gamma)
    tau.y[j] <- pow(sigma.y[j], -2)
```

```
    sigma.y[j] ~ dlnorm(mu.sigma.y, tau.sigma.y)
  }
  tau.gamma <- pow(sigma.gamma, -2)
  sigma.gamma ~ dunif(0, 100)
  mu.sigma.y ~ dnorm(0, .0001)
  tau.sigma.y <- pow(sigma.sigma.y, -2)
  sigma.sigma.y ~ dunif(0, 100)
}
```

I specify the separate error variance for each group beginning in the third line of the code, `y[i] ~ dnorm (y.hat[i], tau.y[group[i]])`. The `tau.y[group[i]]` allows the error variance for the observation to vary by the group of that observation. I then define `tau.y[j] <- pow(sigma.y[j], -2)` for each observation j, where `sigma.y[j] ~ dlnorm(mu.sigma.y, tau.sigma.y)`. I give `sigma.y[j]` a log-normal prior as it is constrained to being greater than zero and the model converges well with this prior. I then specify `mu.sigma.y` and `tau.sigma.y` in the standard form for JAGS.

To run the model, I first load and manipulate the data. This is done in the chunk of code below.

```
# Loading the data from a csv
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()
individual <- 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] <- individual
# 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", "individual.id")

# Making the true DV by taking the absolute value
model.data$error <- abs(model.data$error)
```

Then, I define the necessary values we will read into our JAGS model; define the data in the model; specify reasonable initial values for our algorithm to begin estimation of the parameters. Then, I define the model and save it to the `age.mod` object.

```
# Defining values for JAGS
J <- length(unique(model.data$group.id))
K <- length(unique(model.data$individual.id))
n <- length(model.data[,1])
```

```
y <- model.data$error
group <- model.data$group.id
individual <- model.data$individual.id

# Lists the data that will be contained in our JAGS model
age.data <- list("n","K","J","y","group","individual")

# Function to return list of starting values of algorithm
# For example, rnorm(J) is vector of length J of random numbers from the N(0,1) distribution
# These are assigned to alpha to start the JAGS iterations
# We also do this for betas, mu.a's, sigma.y's, and sigma.a's -- that is, all parameters
age.inits <- function(){list(mu=rnorm(1), gamma=rnorm(J), delta=rnorm(K), sigma.delta=runif(1),
       sigma.sigma.y=runif(1), sigma.y=runif(J), sigma.gamma=runif(1), mu.sigma.y=runif(1))}

# This is vector of names of parameters we want to save from the JAGS run
age.parameters <- c("mu", "gamma", "delta", "sigma.y", "sigma.gamma","sigma.delta")

# The model itself
age.mod <- 'model{
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y[group[i]])
    y.hat[i] <- mu + gamma[group[i]] + delta[individual[i]]
  }
  mu ~ dnorm(0, .0001)

  for (k in 1:K){
    delta[k] ~ dnorm(0, tau.delta)
    }
  tau.delta <- pow(sigma.delta, -2)
  sigma.delta ~ dunif(0, 100)

  for(j in 1:J){
    gamma[j] ~ dnorm(0, tau.gamma)
    tau.y[j] <- pow(sigma.y[j], -2)
    sigma.y[j] ~ dlnorm(mu.sigma.y, tau.sigma.y)
  }
  tau.gamma <- pow(sigma.gamma, -2)
  sigma.gamma ~ dunif(0, 100)
  mu.sigma.y ~ dnorm(0, .0001)
  tau.sigma.y <- pow(sigma.sigma.y, -2)
  sigma.sigma.y ~ dunif(0, 100)
}'
```

Then, I run the model, saving it to the `age` object.I specify that our algorithm should use 3 chains to simulate parameter values, as well as specifying 1000 iterations, with the standard burn-in period of the first half of the iterations.

```
# This is the actual JAGS run
set.seed(100)
age <- jags(data=age.data, inits=age.inits, parameters.to.save=age.parameters,
                model.file=textConnection(age.mod), n.chains=3, n.iter=1000, DIC=F)
Compiling model graph
   Resolving undeclared variables
```

```
   Allocating nodes
Graph information:
   Observed stochastic nodes: 100
   Unobserved stochastic nodes: 35
   Total graph size: 495

Initializing model
```

I present the output from this model in Table 14 below. We see that the separate parameters for groups and individuals vary, as expected – there are expected to be some groups better at guessing ages than others, and some individuals whose ages are easier to guess than others. We see that the estimated across-group standard deviation, $\hat{\sigma_\gamma}$, is much smaller than the estimated across-individual standard deviation, $\hat{\sigma_\delta}$. This makes sense – we expect that groups of people guessing ages are fairly close in their abilities to guess ages, whereas we know that some individuals ages are very difficult to guess. We see that there is some variation across groups in the error variance – the estimates of σ_y vary for each j – suggesting we were right to model this variation.

Looking at the $\hat{R}$ column, we see that the algorithm converged reasonably well – none of these values are above 1.1. Looking at the n_{eff} column suggests that, generally, we have enough effective simulations to achieve precise estimates of our parameters.

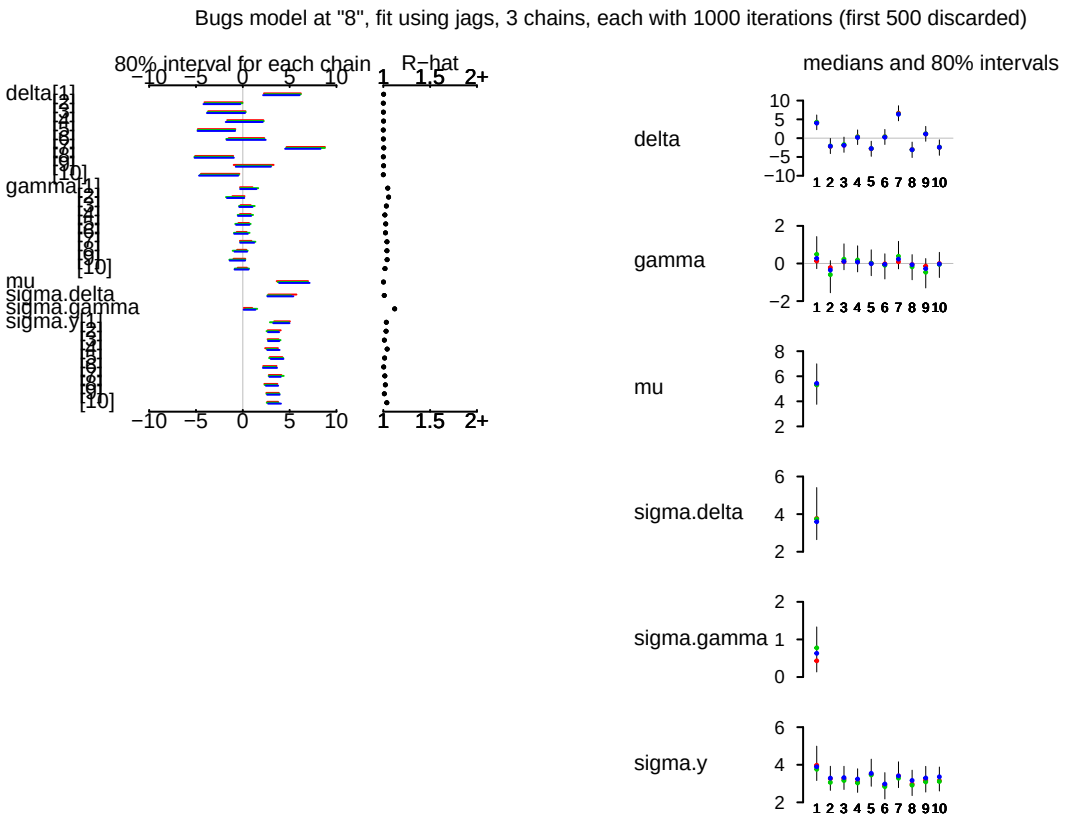
Table 14: Age Guessing Model, Varying Parameters for Group and Individual, Separate Error Variance for each Group

Variable	Mean	Std. Dev.	2.5%	25%	50%	75%	97.5%	$\hat{R}$	n_{eff}
δ_1	4.110	1.686	0.838	3.036	4.097	5.160	7.573	1.000	1500
⋮									
δ_{10}	-2.475	1.672	-5.924	-3.474	-2.463	-1.417	0.600	1.001	1500
γ_1	0.460	0.738	-0.632	-0.022	0.247	0.881	2.288	1.025	150
⋮									
γ_{10}	-0.052	0.572	-1.248	-0.318	-0.011	0.214	1.125	1.013	1500
μ	5.454	1.437	2.671	4.579	5.430	6.346	8.307	1.000	1500
σ_δ	4.037	1.296	2.315	3.159	3.805	4.610	7.193	1.013	160
σ_γ	0.687	0.496	0.024	0.281	0.612	1.017	1.826	1.088	39
σ_{y1}	3.884	0.707	2.954	3.365	3.720	4.268	5.541	1.009	280
⋮									
σ_{y10}	3.216	0.488	2.344	2.909	3.192	3.446	4.405	1.001	1500

DIC info (using the rule, pD = var(deviance)/2): pD = 23.5 and DIC = 543.2

A second method we can use to aid in interpretation of model results is to examine the built-in graphical summary of a JAGS simulation, accessed through the `plot` function. I plot this summary below. We see that the plot provides us similar results to our tabular analysis above.

```
plot(age)
```


Bugs model at "8", fit using jags, 3 chains, each with 1000 iterations (first 500 discarded)
80% interval for each chain
R−hat
medians and 80% intervals
delta[1]
gamma[1]
mu
sigma.delta
sigma.gamma
sigma.y[1]
−10 −5 0 5 10
1 1.5 2+
delta
gamma
mu
sigma.delta
sigma.gamma
sigma.y
1 2 3 4 5 6 7 8 9 10