Introduction to MCMC

Bayesian Psychometric Models, Lecture 7

The current lecture is built loosely off of Levy and Mislevy (2016) Chapter 5, with information from Chapters 4 and 6 (although not necessarily all from the book).

Introduction to Markov Chain Monte Carlo Estimation

Bayesian analysis is all about estimating the posterior distribution

- Up until now, we have worked with posterior distributions that fairly well-known
  - Beta-Binomial had a Beta distribution
  - In general, likelihood distributions from the exponential family have conjugate priors
    - Conjugate prior: the family of the prior is equivalent to the family of the posterior
- Most of the time, however, posterior distributions are not easily obtainable
  - No longer able to use properties of the distribution to estimate parameters
- It is possible to use an optimization algorithm (e.g., Newton-Raphson or Expectation-Maximization) to find maximum value of posterior distribution
  - But, such algorithms may take a very long time for high-dimensional problems
- Instead: "sketch" the posterior by sampling from it – then use that sketch to make inferences
  - Sampling is done via MCMC

Markov Chain Monte Carlo

- MCMC algorithms iteratively sample from the posterior distribution
  - For fairly simplistic models, each iteration has independent samples
  - Most models have some layers of dependency included
    - Can slow down sampling from the posterior
- There are numerous variations of MCMC algorithms
  - Most of these specific algorithms use one of two types of sampling:
    1. Direct sampling from the posterior distribution (i.e. Gibbs sampling)
       - Often used when conjugate priors are specified
    2. Indirect (rejection-based) sampling from the posterior distribution (e.g., Metropolis-Hastings)
- Efficiency is the main reason for so many algorithms
  - Efficiency in this context: How quickly the algorithm converges and provides adequate coverage ("sketching") of the posterior distribution
  - No one algorithm is uniformly most efficient for all models (here model = likelihood $\times$ prior)
- The good news is that many software packages (JAGS and MPlus, especially) don't make you choose which specific algorithm to use
- The bad news is that sometimes your model may take a large amount of time to reach convergence (think days or weeks)
- You can also code your own custom algorithm to make things run smoother

Commonalities Across MCMC Algorithms

- Despite having fairly broad differences regarding how algorithms sample from the posterior distribution, there are quite a few things that are similar across algorithms:
  1. A period of the Markov chain where sampling is not directly from the posterior
     - The burnin period

2. Methods used to assess convergence of the chain to the posterior distribution
    - Often involving the need to use multiple chains with independent and differing starting values
3. Summaries of the posterior distribution

- Further, rejection-based sampling algorithms often need a tuning period to make the sampling more efficient
    - The tuning period comes before the algorithm begins its burnin period

To demonstrate each type of algorithm, we will use a model for a normal distribution (Chapter 4 of Levy & Mislevy, 2016) - We will investigate each, briefly - We will then switch over to JAGS to show the syntax and let JAGS work

We will conclude by talking about assessing convergence and how to report parameter estimates.

Example Data: Post-Diet Weights

Borrowing an example from (https://stats.idre.ucla.edu/spss/library/spss-libraryhow-do-i-handle-interactions-of-continuous-an //stats.idre.ucla.edu/spss/library/spss-libraryhow-do-i-handle-interactions-of-continuous-andcategorical-variables/], the file DietData.csv contains data from 30 respondents who participated in a study regarding the effectiveness of three types of diets. Variables in the data set are:

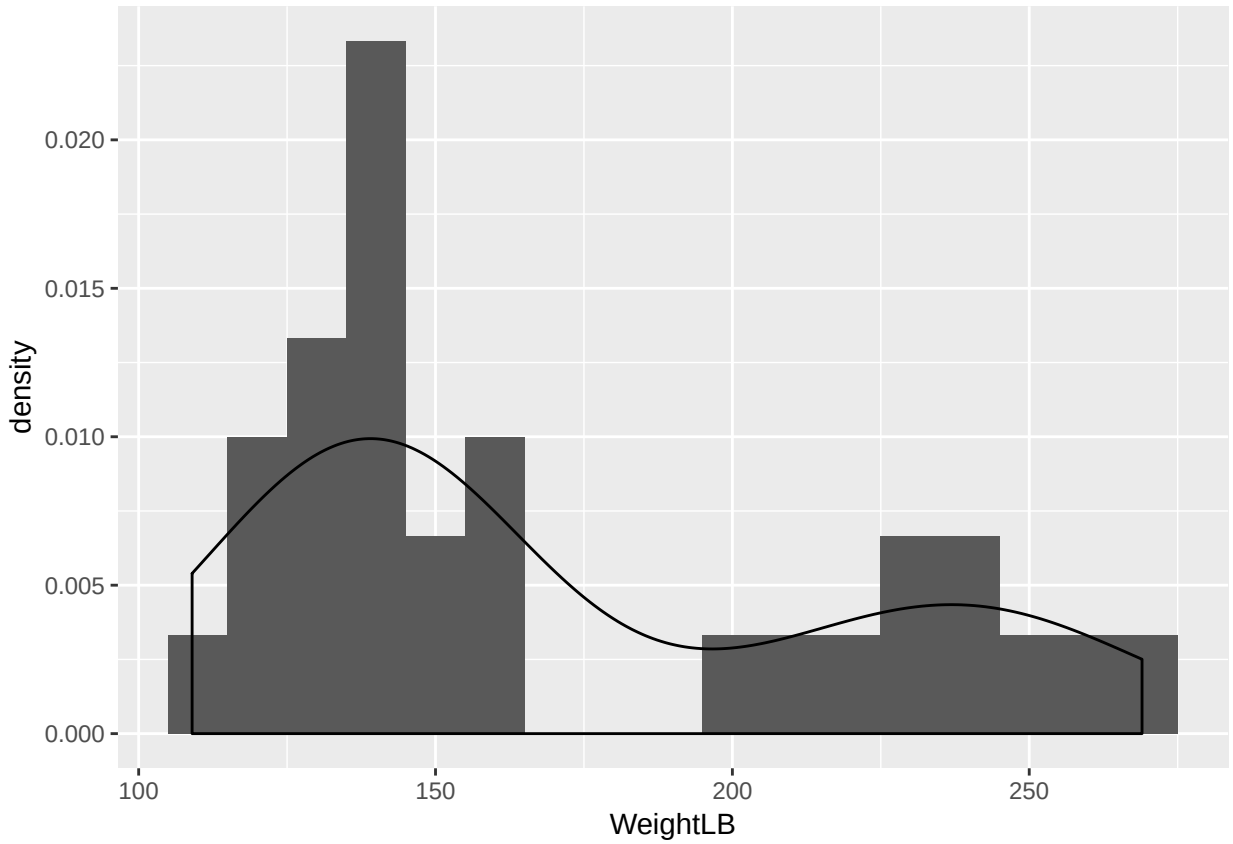
1. Respondent: Respondent number 1-30
2. DietGroup: A 1, 2, or 3 representing the group to which a respondent was assigned
3. HeightIN: The respondent's height in inches
4. WeightLB: The respondent's weight, in pounds, recorded following the study

Here, weight is the dependent variable. The motivating research question is:

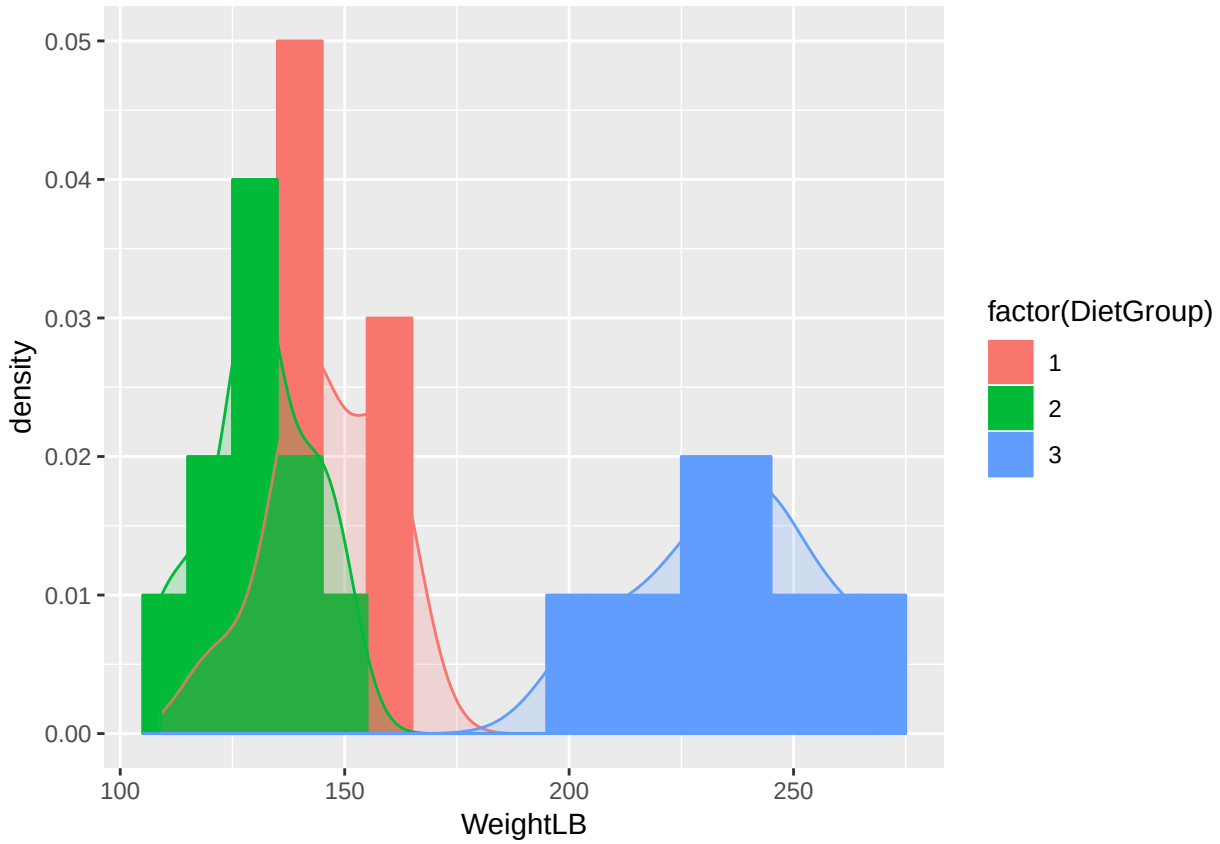
> Are there differences in final weights between the three diet groups, and, if so, what are the nature of the differences?

The following syntax reads the data into R and provides some pictures of the data:

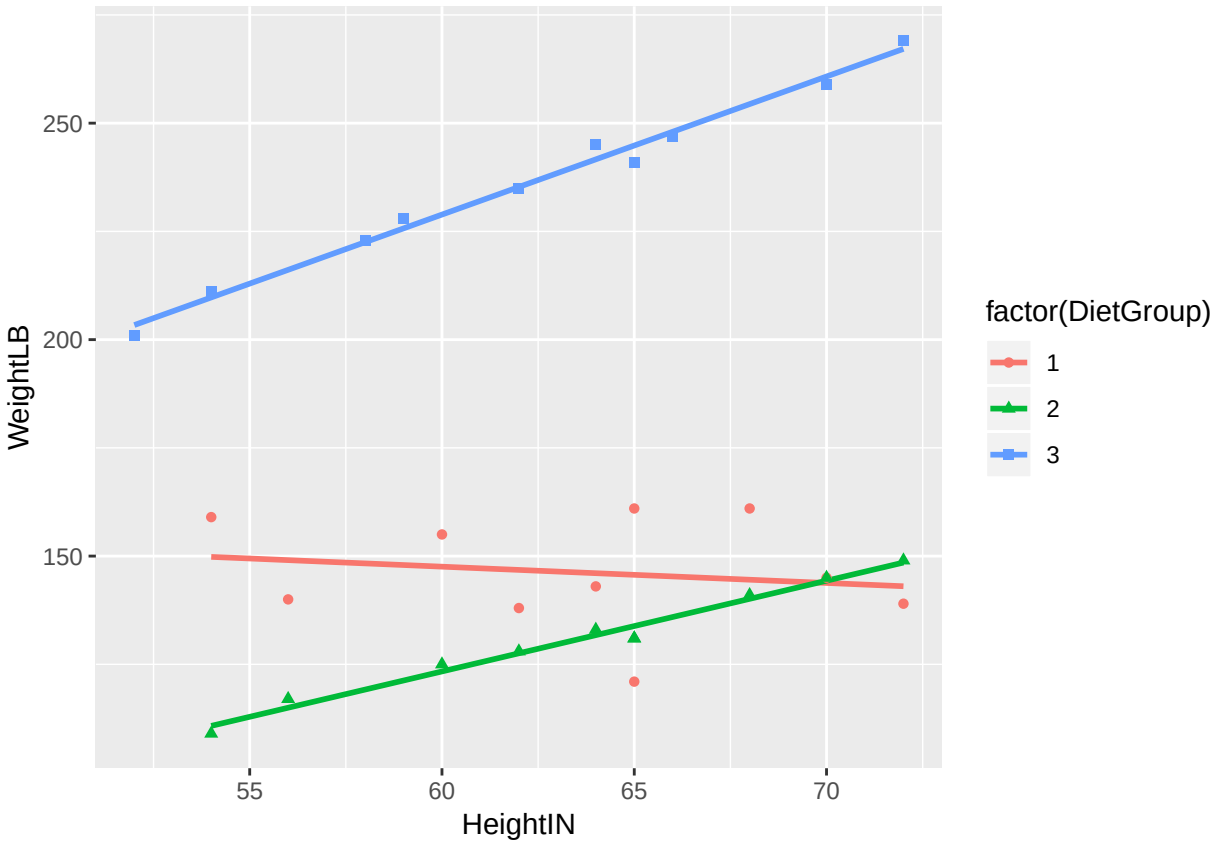
```
DietData = read.csv(file = "DietData.csv")

ggplot(data = DietData, aes(x = WeightLB)) +
  geom_histogram(aes(y = ..density..), position = "identity", binwidth = 10) +
  geom_density(alpha=.2)
```




```
ggplot(data = DietData, aes(x = WeightLB, color = factor(DietGroup), fill = factor(DietGroup))) +
  geom_histogram(aes(y = ..density..), position = "identity", binwidth = 10) +
  geom_density(alpha=.2)
```



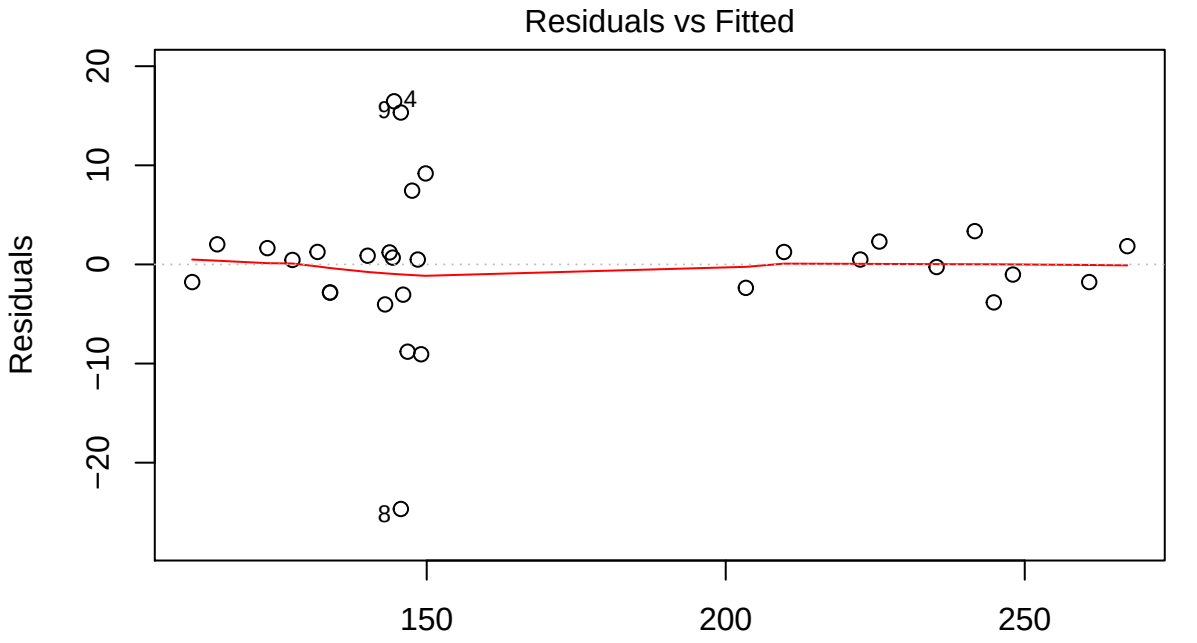

```
ggplot(data = DietData, aes(x = HeightIN, y = WeightLB, shape = factor(DietGroup), color = factor(DietG
  geom_smooth(method = "lm", se = FALSE) + geom_point()
```

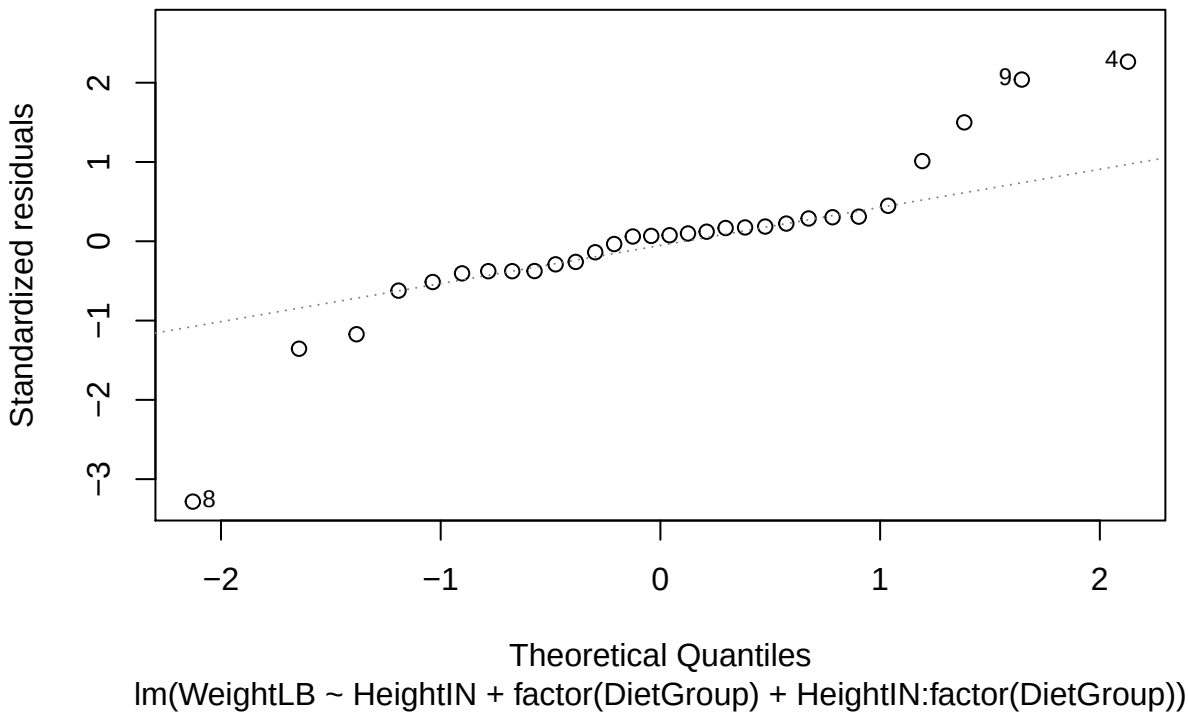


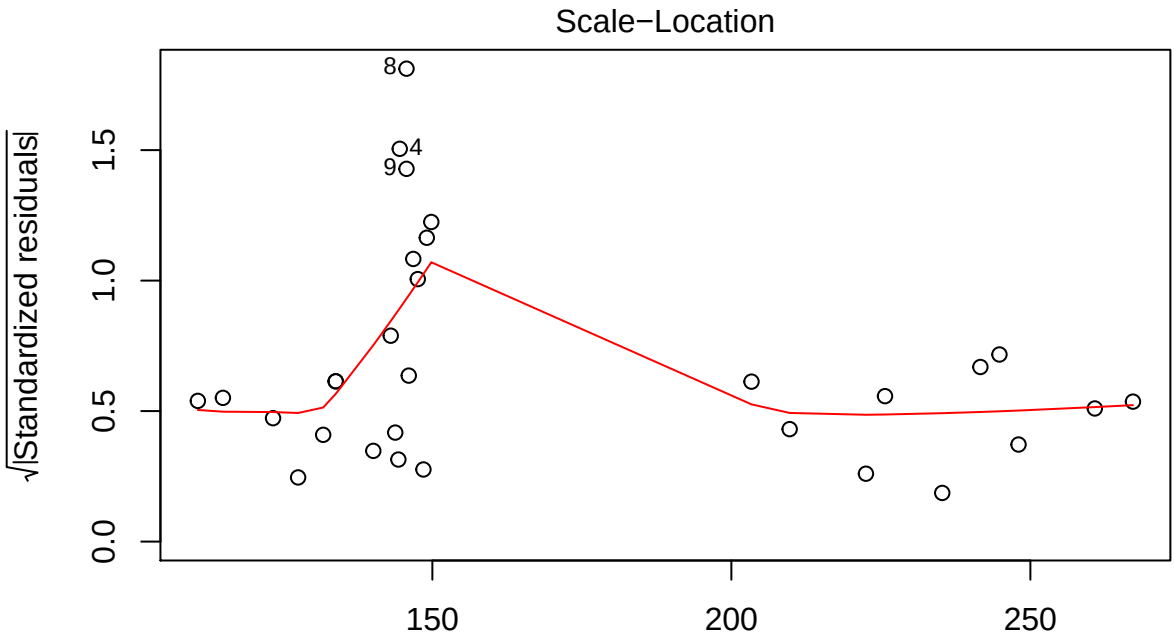
Now, your turn to answer questions:

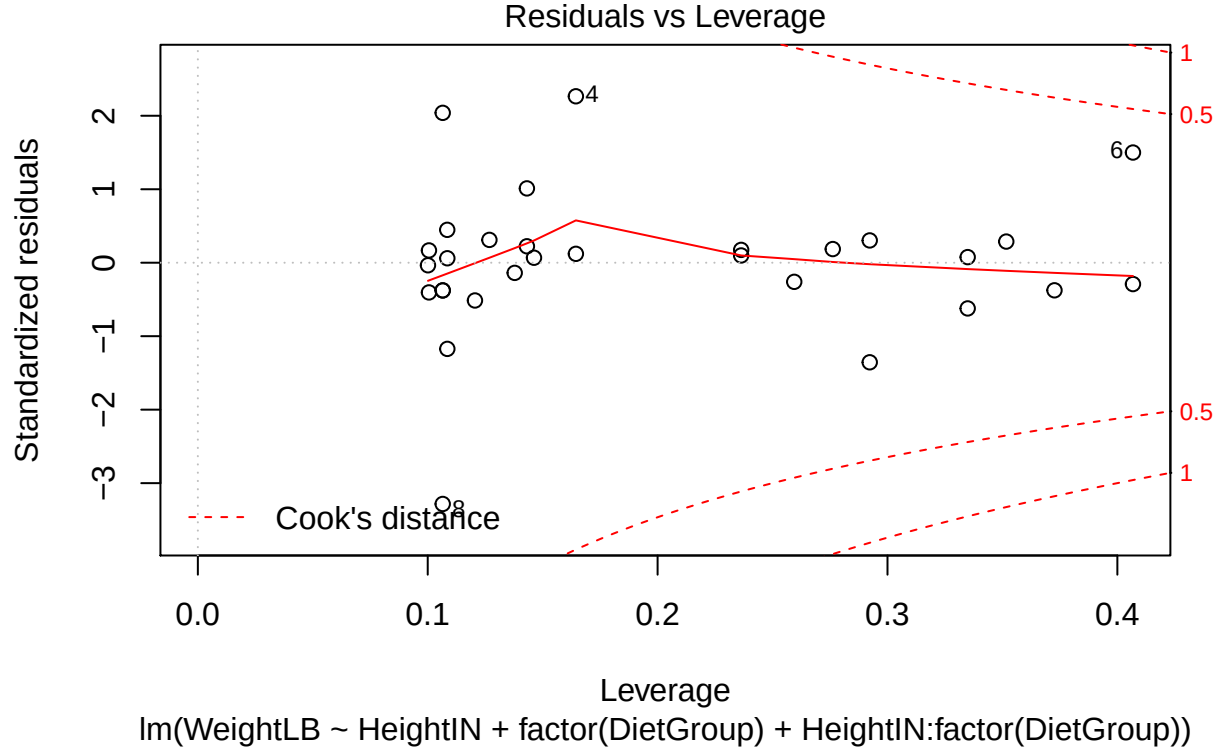
1. What type of analysis seems most appropriate for these data?
2. Is the dependent variable (`WeightLB`) is appropriate as-is for such analysis or does it need transformed?

```
# full analysis model suggested by data:
FullModel = lm(formula = WeightLB ~ HeightIN + factor(DietGroup) + HeightIN:factor(DietGroup), data = D

# examining assumptions and leverage of fit
plot(FullModel)
```


Residuals vs Fitted
Residuals
20
10
0
−10
−20
9
4
8
150
200
250
Fitted values
lm(WeightLB ~ HeightIN + factor(DietGroup) + HeightIN:factor(DietGroup))


Normal Q−Q
Standardized residuals
2
1
0
−1
−2
−3
4
9
8
−2
−1
0
1
2
Theoretical Quantiles
lm(WeightLB ~ HeightIN + factor(DietGroup) + HeightIN:factor(DietGroup))






```
# looking at ANOVA table
anova(FullModel)
```

```
## Analysis of Variance Table
##
## Response: WeightLB
##                            Df Sum Sq Mean Sq F value    Pr(>F)
```

```
## HeightIN                     1    684     684  10.819  0.003092 **
## factor(DietGroup)            2  66726   33363 527.950 < 2.2e-16 ***
## HeightIN:factor(DietGroup)   2   2186    1093  17.293 2.234e-05 ***
## Residuals                   24   1517      63
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# looking at parameter summary
summary(FullModel)
```

```
##
## Call:
## lm(formula = WeightLB ~ HeightIN + factor(DietGroup) + HeightIN:factor(DietGroup),
##     data = DietData)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.6724  -2.7169   0.4958   1.7918  16.4581
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   170.1664    29.2786   5.812 5.42e-06 ***
## HeightIN                       -0.3768     0.4587  -0.822 0.419392
## factor(DietGroup)2           -172.5639    41.4062  -4.168 0.000345 ***
## factor(DietGroup)3           -132.6675    38.7846  -3.421 0.002241 **
## HeightIN:factor(DietGroup)2    2.4727     0.6486   3.812 0.000846 ***
## HeightIN:factor(DietGroup)3    3.5666     0.6132   5.817 5.36e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.949 on 24 degrees of freedom
## Multiple R-squared:  0.9787, Adjusted R-squared:  0.9742
## F-statistic: 220.3 on 5 and 24 DF,  p-value: < 2.2e-16
```

Bayesian Analysis of Our Example Data

Let's examine how our example data should be analyzed using Bayesian methods. To do so we need to specify:

1. Likelihood function (from the model)
2. Prior distributions for all model parameters

Linear Models Likelihood Function (Scalar Version)

The classic linear model is one where, for a respondent r ($r = 1, \ldots, N$), a dependent variable (or outcome), y_r, is predicted via a set of independent variables (or predictors), x_{rv} where $v = 1, \ldots, V$ and, sometimes, their interactive product, by means of a set of regression coefficients β_v:

$$Y_r = \beta_0 + \beta_1 x_{r1} + \beta_2 x_{r2} + \cdots + \beta_V x_{rV} + e_r = \mathbf{x}_r \boldsymbol{\beta} + e_r$$

Here, e_r is the residual or error term for respondent r, with $e_r \sim N(0, \sigma_e^2)$.

So, loosely, we can say that an observation $Y_r \sim N(\mathbf{x}_r \boldsymbol{\beta}, \sigma_e^2)$. This now gives us the set of parameters for which we need to specify prior distributions: $\boldsymbol{\beta}$ and σ_e^2.

Linear Models: Conjugate Prior Distributions

Conjugate priors are prior distributions that have the same form as the posterior distribution. As we will see in another lecture, conjugate priors are helpful because they make our Bayesian sampling algorithms run efficiently. We will stay with conjugate prior distributions for this lecture.

Choosing conjugate prior distributions in Bayesian linear models has one complicating factor: The likelihood for $\boldsymbol{\beta}$ is conditional on σ_e^2. So, let's start there. In particular:

$$f\left(\boldsymbol{\beta}, \sigma_e^2\right) = f\left(\boldsymbol{\beta} \mid \sigma_e^2\right) f\left(\sigma_e^2\right)$$

Prior for σ_e^2

To make sense of this, we can start with σ_e^2, which has a conjugate prior distribution that is inverse-gamma. The inverse-gamma distribution (see (https://en.wikipedia.org/wiki/Inverse-gamma_distribution)[https://en.wikipedia.org/wiki/Inverse-gamma_distribution]) is one that has support of $(0, \infty]$, which matches the range of σ_e^2. The distribution has two parameters, α (sometimes called the shape; $\alpha > 0$) and β (sometimes called the scale, although in the `invgamma` R package, it is the rate, $\beta > 0$).

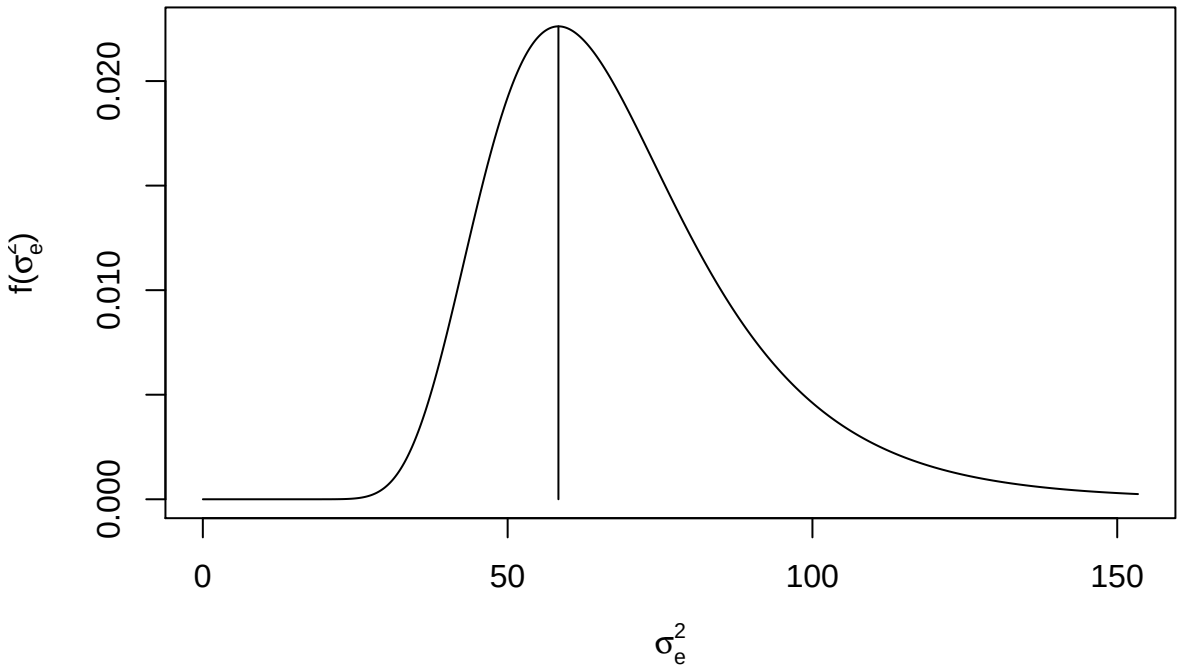
To get a sense of how these parameters function, the mode is a good measure of central tendency, which is $\frac{\beta}{\alpha+1}$. In practice, we can rephrase these in terms of our expected degrees of freedom total ν_0 and our expected residual variance σ_{e0}^2:

$$\alpha_0 = \frac{\nu_0}{2}$$

$$\beta_0 = \frac{\nu_0 \sigma_0^2}{2}$$

In the plot below, I use the values from our classical linear model analysis to provide an example of what the prior looks like:

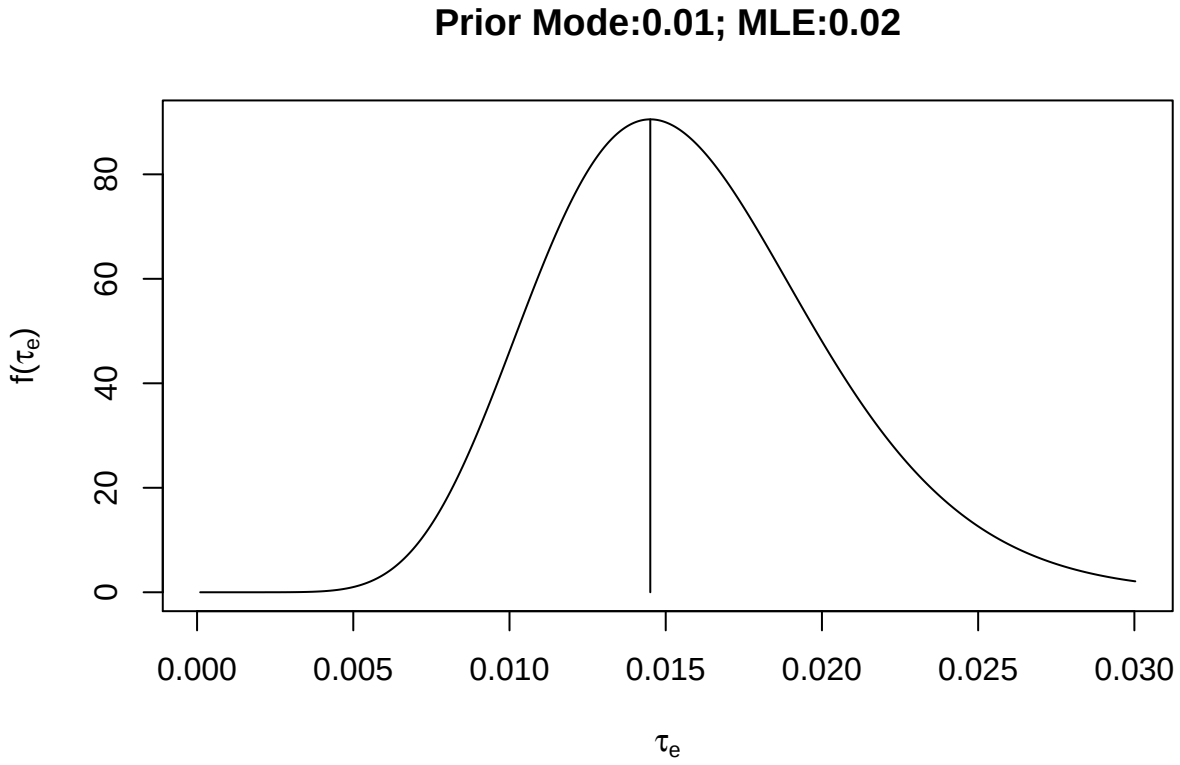
```
nu.0 = summary(FullModel)$df[2]
sigma2.0 = summary(FullModel)$sigma^2

alpha.0 = nu.0/2
beta.0 = nu.0*sigma2.0/2

PriorMode = beta.0/(alpha.0+1)

maxX = qinvgamma(.995, shape = alpha.0, rate= beta.0)
minX = 0.001
step = (maxX - minX)/10000

sigma2 = seq(minX, maxX, step)
height = dinvgamma(x = sigma2, shape = alpha.0, rate = beta.0)

plot(x = sigma2, y = height, type = "l", ylab = expression(paste("f(", sigma[e]^2, ")")), xlab = expres
     main = paste0("Prior Mode:", round(PriorMode, 2), "; MLE:", round(sigma2.0, 2)))

lines(x = c(PriorMode, PriorMode), y = c(0, dinvgamma(x = PriorMode, shape = alpha.0, rate = beta.0)))
```



One complicating factor is that for normal distributions (where we will need to put the parameter σ_e^2 with this prior) JAGS uses a "precision" parameter of $\frac{1}{\sigma_e^2}$ rather than a "dispersion" of σ_e^2. Therefore, we must specify our prior for $\tau_e = \frac{1}{\sigma_e^2}$, which follows a gamma distribution.

The good news is that the parameters of the gamma distribution are the same, however, you have to check what they are called in R. Here is how the previous plot looks with a gamma distribution:

```
nu.0 = summary(FullModel)$df[2]
sigma2.0 = summary(FullModel)$sigma^2
tau.e.0 = 1/sigma2.0

alpha.0 = nu.0/2
beta.0 = nu.0*sigma2.0/2

PriorMode = (alpha.0-1)*(1/beta.0)

maxX = qgamma(.995, shape = alpha.0, rate= beta.0)
minX = 0.0001
step = .00001

tau.e = seq(minX, maxX, step)
height = dgamma(x = tau.e, shape = alpha.0, rate = beta.0)

plot(x = tau.e, y = height, type = "l", ylab = expression(paste("f(", tau[e], ")")), xlab = expression(
     main = paste0("Prior Mode:", round(PriorMode, 2), "; MLE:", round(1/sigma2.0, 2)))
lines(x = c(PriorMode, PriorMode), y = c(0, dgamma(x = PriorMode, shape = alpha.0, rate = beta.0)))
```



"Uninformative" Choices of Conjugate Prior for σ_e^2 (really, τ_e)

One way to make an "uninformative" but conjugate prior for σ_e^2 is to recall the terms we used to specify the prior parameters: ν_0 as the expected DF total and σ_{e0}^2 as the expected residual variance.

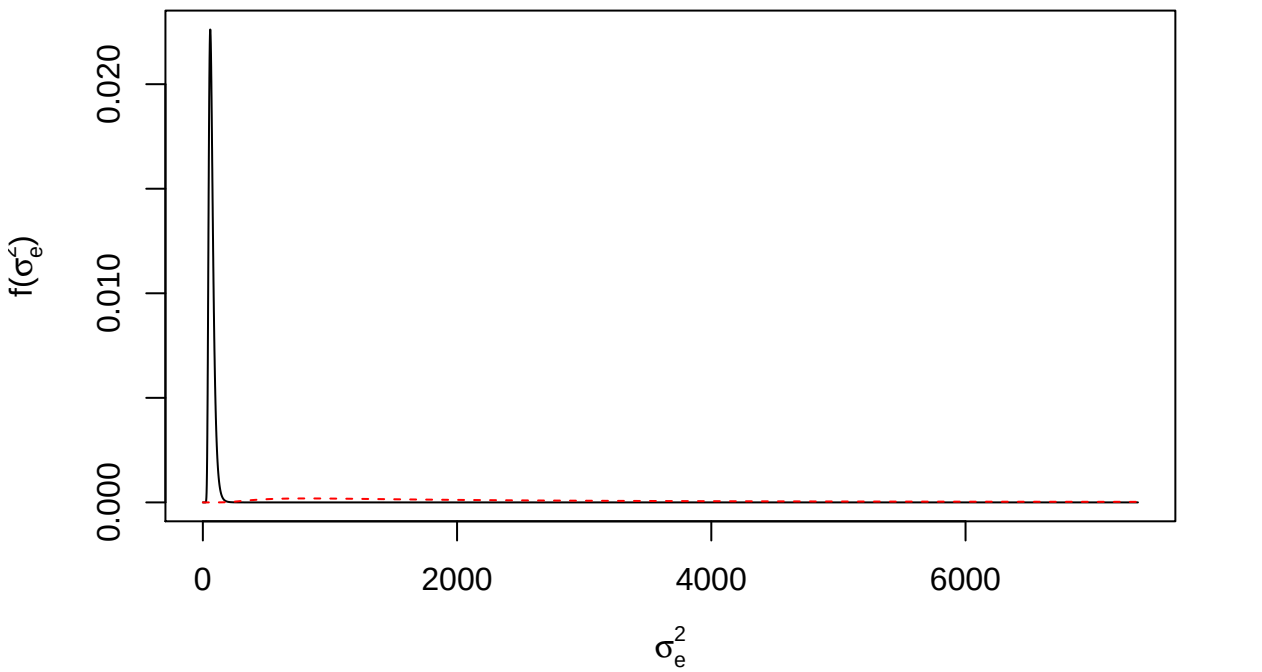
For the residual variance: 1. The maximum residual variance is the marginal variance of y: occurs when $\boldsymbol{\beta} = \mathbf{0}$ 2. Higher values of variance parameters result in higher posterior standard deviations for $\boldsymbol{\beta}$, meaning they are more conservative

So, we'll start with an estimate of $\sigma_{e0}^2 = \sigma_y^2$. For our sample, $\sigma_y^2 = 2452.14$.

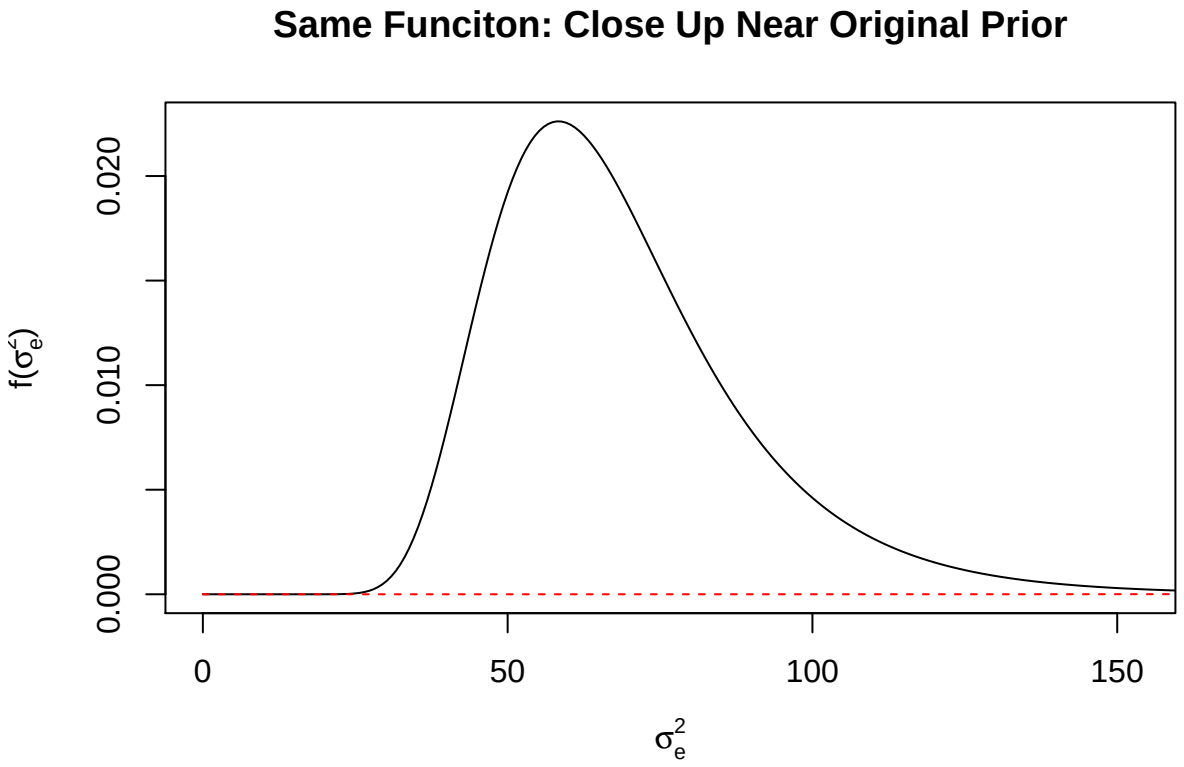
Next, recall that ν_0 is the expected degrees of freedom total for the analysis. As degrees of freedom decrease, the asymptotic sampling error of σ_e^2 gets large, so you can pick a value of ν_0 that is small (but above 0). See the following plot (for σ_e^2, but the parameter choices apply to τ):

```
sigma2.0.U = var(DietData$WeightLB)
nu.0.U = 1

alpha.0.U = nu.0.U/2
beta.0.U = (nu.0.U*sigma2.0.U)/2

nu.0 = summary(FullModel)$df[2]
sigma2.0 = summary(FullModel)$sigma^2

alpha.0 = nu.0/2
beta.0 = (nu.0*sigma2.0)/2

PriorMode = beta.0/(alpha.0+1)

maxX = 3*sigma2.0.U
minX = 0.001
```

```r
step = (maxX - minX)/10000

sigma2 = seq(minX, maxX, step)

height = dinvgamma(x = sigma2, shape = alpha.0, rate = beta.0)
heightU = dinvgamma(x = sigma2, shape = alpha.0.U, rate = beta.0.U)

matplot(
  x = cbind(sigma2, sigma2),
  y = cbind(height, heightU),
  type = "l",
  ylab = expression(paste("f(", sigma[e] ^ 2, ")")),
  xlab = expression(sigma[e] ^ 2),
  main = "Red is uninformative prior; black is prior based on suggestion in book"
)
```




```r
matplot(
  x = cbind(sigma2, sigma2),
  y = cbind(height, heightU),
  type = "l",
  ylab = expression(paste("f(", sigma[e] ^ 2, ")")),
  xlab = expression(sigma[e] ^ 2),
  main = "Same Funciton: Close Up Near Original Prior",
  xlim = c(0, qinvgamma(.995, shape = alpha.0, rate = beta.0))
)
```



Prior for $\boldsymbol{\beta}$

Conditional on a value of of σ_e^2, the conjugate prior for $\boldsymbol{\beta}$ follows a multivariate normal distribution: $\boldsymbol{\beta} \sim MVN\left(\bar{\boldsymbol{\beta}}, \boldsymbol{\Sigma}_{\boldsymbol{\beta}}\right)$. Often, this is expressed as a set of independent univariate priors where the mean for each β is given along with some type of variance. That said, the use of normal priors in JAGS uses precision instead of variance (so 1/variance).

These priors can be made to be uninformative by selecting very low values for precision (meaning very high levels of variance). Remember, 99% of the mass of a univariate normal distribution falls within +/- 3SD.

That said, the scale of the regression coefficients will not be constant, so if picking one value for a hyperparameter for the prior of each, pick a very large variance

Bayesian Linear Models Analysis with JAGS

The first step to building our Bayesian linear model analysis in JAGS is to put all of our data and prior values into an R list object. Here, we will

```
# specify specs for analysis
n.chains = 4

# specify values of hyperparameters for prior distributions

# for sigma2_e (tau)
sigma2.0 = var(DietData$WeightLB)
nu.0 = 1

alpha.tau.0 = nu.0/2
beta.tau.0 = (nu.0*sigma2.0)/2
```

```r
# for betas:
betaMean.0 = 0
betaVariance.0 = 100000000
betaTau.0 = 1/betaVariance.0

# take data frame and create model matrix -- which gives us the columns of X:
BayesianFullModel.Matrix = model.matrix(object = FullModel)

# append with weight and then select only columns needed for model:
BayesianFullModel.Matrix = data.frame(cbind(DietData$WeightLB, BayesianFullModel.Matrix))
names(BayesianFullModel.Matrix) = c("weight", "intercept", "height", "group2", "group3", "hg2", "hg3")
BayesianFullModel.Matrix = BayesianFullModel.Matrix[c("weight", "height", "group2", "group3")]

# add data and N to list object that we will pass to JAGS
BayesianFullModel.JAGS.Data = list(N = nrow(DietData),
                                   weight = BayesianFullModel.Matrix$weight,
                                   height = BayesianFullModel.Matrix$height,
                                   group2 = BayesianFullModel.Matrix$group2,
                                   group3 = BayesianFullModel.Matrix$group3,
                                   betaMean.0 = betaMean.0,
                                   betaTau.0 = betaTau.0,
                                   alpha.tau.0 = alpha.tau.0,
                                   beta.tau.0 = beta.tau.0)
```

Next, we can create our JAGS model syntax. We will use the `R2jags` package, so we enclose the syntax in a function. Here, we put the names of the variables and hyperparameters we created in our previous step directly into the JAGS model syntax.

There are several things to note here:

- Hyperparameter values are explicit numbers in syntax, not variables entered into JAGS
- The example syntax on p. 129 has a calculation of R^2 – however, it uses the MLE for σ_y^2 and a version of the MLE for σ_e^2, so it is not included

```r
BayesianFullModel.Syntax = "

  model{

    # prior distributions:
    #   Note that terms on the left are model parameter names we create here
    beta.0             ~ dnorm(betaMean.0, betaTau.0) # prior for the intercept
    beta.height        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional slope of height
    beta.group2        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwee
    beta.group3        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwee
    beta.height.group2 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 2
    beta.height.group3 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 3
    tau.e              ~ dgamma(alpha.tau.0, beta.tau.0)      #prior for 1/sigma^2_e, residual precisio


    # conditional distribution of the data (uses priors above)
    for (i in 1:N){

      # creating conditional mean to put into model statement
      weight.hat[i] <- beta.0 + beta.height*height[i] + beta.group2*group2[i] + beta.group3*group3[i] +
```

```
      # error values (for R^2)
      error[i] = weight[i]-weight.hat[i]

      # likelihood from model:
      weight[i] ~ dnorm(weight.hat[i], tau.e)

    }

    # created values to track throughout the Markov chain
    sigma2.e  <- 1/tau.e                  #create residual variance from 1/precision
    sigma.e   <- sqrt(sigma2.e)           #create residual SD

    # calculate mean difference between groups 2 and 3:
    dif.group2.group3 <- beta.group3 - beta.group2

    # calculate R^2 based on http://www.stat.columbia.edu/~gelman/research/unpublished/bayes_R2.pdf
    variance.pred = sd(weight.hat[])*sd(weight.hat[])
    variance.error = sd(error[])*sd(error[])
    Rsquare = variance.pred/(variance.pred + variance.error)
  }

"
# the parameter names come from the model syntax in the previous step
BayesianFullModel.Parameters = c("beta.0",
                                 "beta.height",
                                 "beta.group2",
                                 "beta.group3",
                                 "beta.height.group2",
                                 "beta.height.group3",
                                 "dif.group2.group3",
                                 "tau.e",
                                 "sigma2.e",
                                 "sigma.e",
                                 "Rsquare",
                                 "deviance")

# in order to have reproducable Markov chains, we need to initialize the random number generator and se
# pick a number: Here is the date I created this syntax
BayesianFullModel.Seed = 06022019

#   Note: here, the random number seed cannot be the same per seed or the chains will be the same
RNGname = c("Wichmann-Hill","Marsaglia-Multicarry", "Super-Duper", "Mersenne-Twister")
if (RNGname[1] %in% c("Wichmann-Hill", "Marsaglia-Multicarry",
  "Super-Duper", "Mersenne-Twister")) {
  RNGname[1] <- paste("base::", RNGname[1], sep = "")
}

BayesianFullModel.Init.Values <- vector("list", n.chains)
for (i in 1:n.chains) {
  BayesianFullModel.Init.Values[[i]]$.RNG.name <- RNGname[1]
  BayesianFullModel.Init.Values[[i]]$.RNG.seed <- BayesianFullModel.Seed + i
}
```

Next, to run JAGS, we need a few more values to save: (1) the names of the parameters we wish to have

JAGS track across the chains and (2) a random number seed for reproducability. Once we have that, we can run JAGS. Below, I will use the `jags()` function, to which we should also provide the number of chains, number of iterations total, and number of burnin iterations.

```
# load two JAGS modules: GLM (which makes the analysis go more efficiently) and DIC (which gives an ind
load.module("glm")
```

```
## module glm loaded
```

```
load.module("dic")
```

```
## module dic loaded
```

```
# Submit model to JAGS for compiling. We'll turn adaptation off to control it in the next line of synta
BayesianFullModel.JAGS = jags.model(file = textConnection(BayesianFullModel.Syntax), data = BayesianFull
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 7
##    Total graph size: 259
##
## Initializing model
```

```
# adapting the model: No adaptation needed as Gibbs Sampling is being used -- but good to try in case a
adapt(object = BayesianFullModel.JAGS, n.iter = 1000)
```

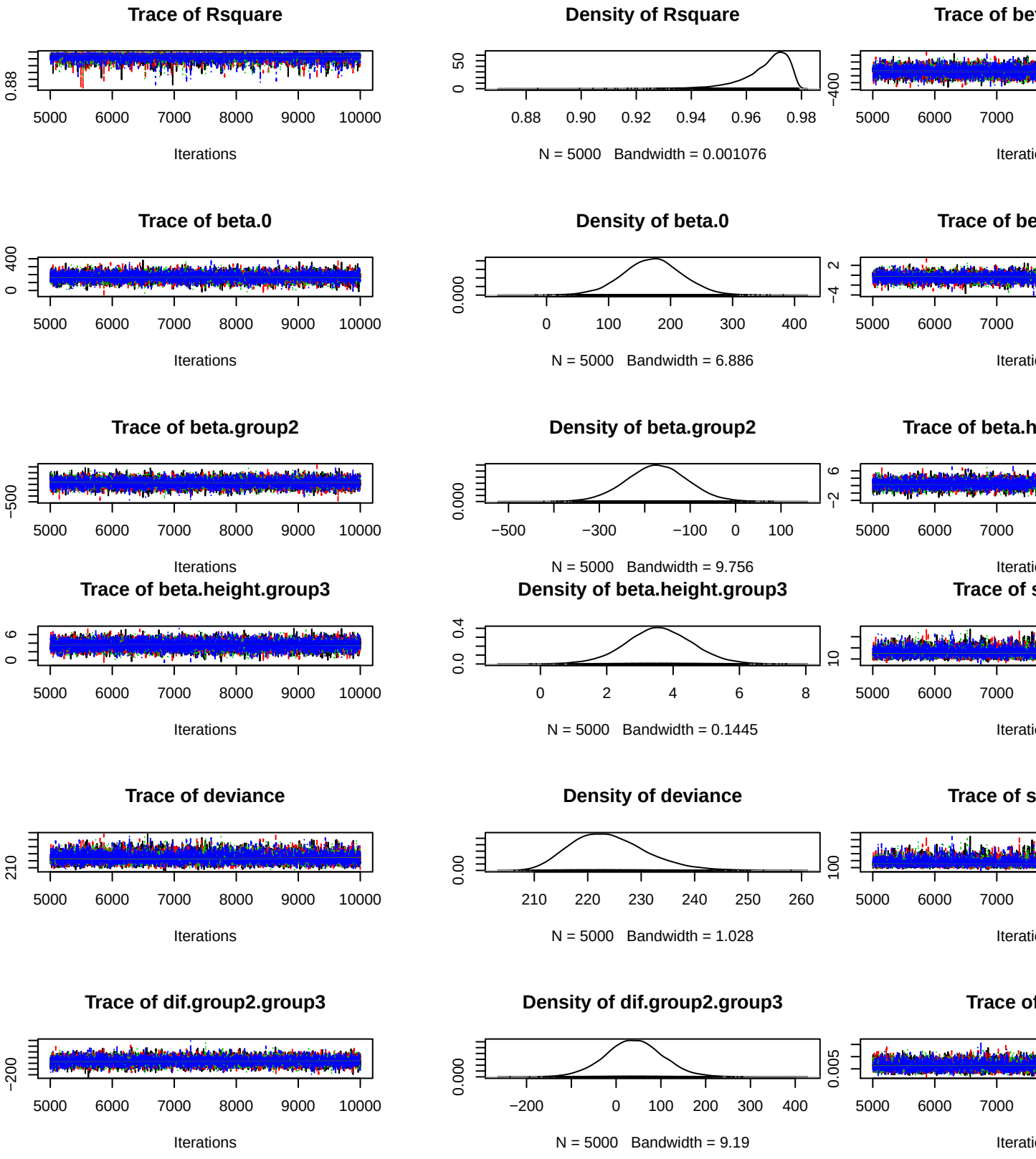
```
## [1] TRUE
```

```
# Draw samples; NOTE: BURNIN IS INCUDED IN SAMPLES
BayesianFullModel.Samples = coda.samples(model = BayesianFullModel.JAGS, variable.names = BayesianFullM

# Remove burnin from samples:
BayesianFullModel.Posterior = window(x = BayesianFullModel.Samples, start = 5001, end = 10000)
```

Checking Chain Convergence

After the model runs, the next step is to assess convergence of the chains to the posterior distribution. This can be accomplished in several ways:

```
# assessing convergence:

# visual assessment
plot(BayesianFullModel.Posterior)
```




```
# using convegence diagnostics:
gelman.diag(BayesianFullModel.Posterior, multivariate = FALSE)
```

```
## Potential scale reduction factors:
##
##                     Point est. Upper C.I.
## Rsquare                      1          1
## beta.0                       1          1
## beta.group2                  1          1
## beta.group3                  1          1
## beta.height                  1          1
## beta.height.group2           1          1
## beta.height.group3           1          1
## deviance                     1          1
## dif.group2.group3            1          1
## sigma.e                      1          1
## sigma2.e                     1          1
## tau.e                        1          1
```

```
# checking autocorrelation: very low for lag 1
autocorr.diag(BayesianFullModel.Posterior)
```

```
##             Rsquare        beta.0  beta.group2 beta.group3  beta.height
## Lag 0   1.000000000  1.0000000000  1.000000000 1.000000000  1.000000000
## Lag 1   0.150625821 -0.0008196497 -0.003412160 0.007992317 -0.001429560
## Lag 5  -0.005211171  0.0098458295 -0.001800741 0.008898029  0.007988504
## Lag 10  0.004830756  0.0212639457 -0.000290625 0.004787050  0.021272860
## Lag 50 -0.005125523  0.0010910225  0.012951877 0.001860613  0.002353988
##        beta.height.group2 beta.height.group3      deviance
## Lag 0        1.0000000000        1.000000000  1.0000000000
## Lag 1       -0.0029447741        0.007692403  0.3234220496
## Lag 5       -0.0024587236        0.007174552  0.0002490533
## Lag 10      -0.0003975335        0.005750982  0.0089525877
## Lag 50       0.0131368090        0.002119942 -0.0044091911
##        dif.group2.group3       sigma.e      sigma2.e        tau.e
## Lag 0        1.000000000  1.0000000000  1.0000000000  1.000000000
## Lag 1        0.004809194  0.2023938463  0.2020213503  0.184900242
## Lag 5       -0.008748620 -0.0033246525 -0.0018487243 -0.007869463
## Lag 10       0.002598268  0.0035326988  0.0012221570  0.008525871
## Lag 50       0.003643658 -0.0004026842  0.0003452465 -0.001911239
```

```
summary(BayesianFullModel.Posterior)
```

```
##
## Iterations = 5001:10000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                          Mean        SD  Naive SE Time-series SE
## Rsquare              9.671e-01  0.009067 6.412e-05      7.474e-05
## beta.0               1.708e+02 48.385970 3.421e-01      3.472e-01
## beta.group2         -1.732e+02 67.877682 4.800e-01      4.723e-01
## beta.group3         -1.333e+02 64.021723 4.527e-01      4.471e-01
## beta.height         -3.862e-01  0.757297 5.355e-03      5.431e-03
```

```
## beta.height.group2  2.482e+00  1.062998 7.517e-03      7.469e-03
## beta.height.group3  3.577e+00  1.012075 7.156e-03      6.983e-03
## deviance            2.240e+02  7.032384 4.973e-02      6.654e-02
## dif.group2.group3   3.987e+01 64.521135 4.562e-01      4.605e-01
## sigma.e             1.301e+01  1.932803 1.367e-02      1.679e-02
## sigma2.e            1.730e+02 53.436527 3.779e-01      4.640e-01
## tau.e               6.285e-03  0.001783 1.261e-05      1.512e-05
##
## 2. Quantiles for each variable:
##
##                           2.5%        25%        50%        75%     97.5%
## Rsquare             9.436e-01    0.96342    0.96937  9.733e-01   0.97735
## beta.0              7.393e+01  139.01303  171.02777  2.021e+02 265.44476
## beta.group2        -3.071e+02 -217.85516 -173.00514 -1.285e+02 -40.27762
## beta.group3        -2.596e+02 -175.50529 -132.88512 -9.171e+01  -5.40781
## beta.height        -1.871e+00   -0.87749   -0.38744  1.079e-01   1.12032
## beta.height.group2  3.910e-01    1.78268    2.48167  3.184e+00   4.58555
## beta.height.group3  1.562e+00    2.91743    3.57406  4.242e+00   5.57869
## deviance            2.123e+02  218.93856  223.38443  2.284e+02 239.68384
## dif.group2.group3  -8.691e+01   -2.51107   39.31963  8.169e+01 168.75507
## sigma.e             9.873e+00   11.62916   12.79293  1.413e+01  17.43055
## sigma2.e            9.748e+01  135.23742  163.65905  1.996e+02 303.82395
## tau.e               3.291e-03    0.00501    0.00611  7.394e-03   0.01026
```

The most common method used for assessing convergence is the Gelman-Rubin $\hat{R}$, which compares the within- and between-chain variance of multiple chains. The `coda` package has a number of convergence diagnostics.

Checking Model-Data Fit: Posterior Predictive Model Checks

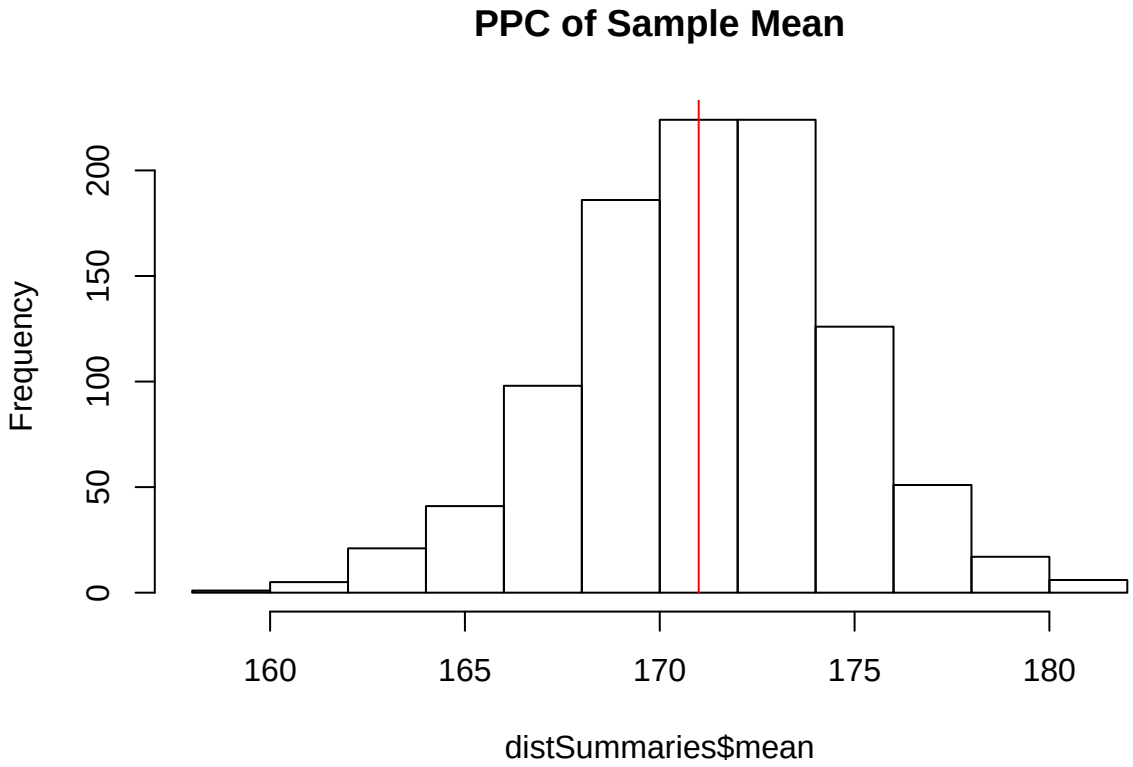
A commonly-used method for evaluating the fit of a Bayesian model to the data is to use posterior predicitive model checks. Posterior predictive model checking is a process that simulates data using model parameters from the converged section of the Markov chain. The Markov chain is important to use, as some parameters have dependencies in the posterior distribution.

```
nSimData = 1000

# create one big data object to randomly sample from:
BayesianFullModel.Posterior.All = do.call("rbind", BayesianFullModel.Posterior)
parameterNames = colnames(BayesianFullModel.Posterior.All)

# set up model matrix to use in repeated samples
BayesianFullModel.Matrix2 = BayesianFullModel.Matrix[,2:4]

# add columns for intercept and interactions
BayesianFullModel.Matrix2 = as.matrix(data.frame(intercept = matrix(data = 1, nrow = nrow(BayesianFullM
                                      BayesianFullModel.Matrix2,
                                      height.group2 = BayesianFullModel.Matrix$height*BayesianFullModel
                                      height.group3 = BayesianFullModel.Matrix$height*BayesianFullModel
colnames(BayesianFullModel.Matrix2)
```

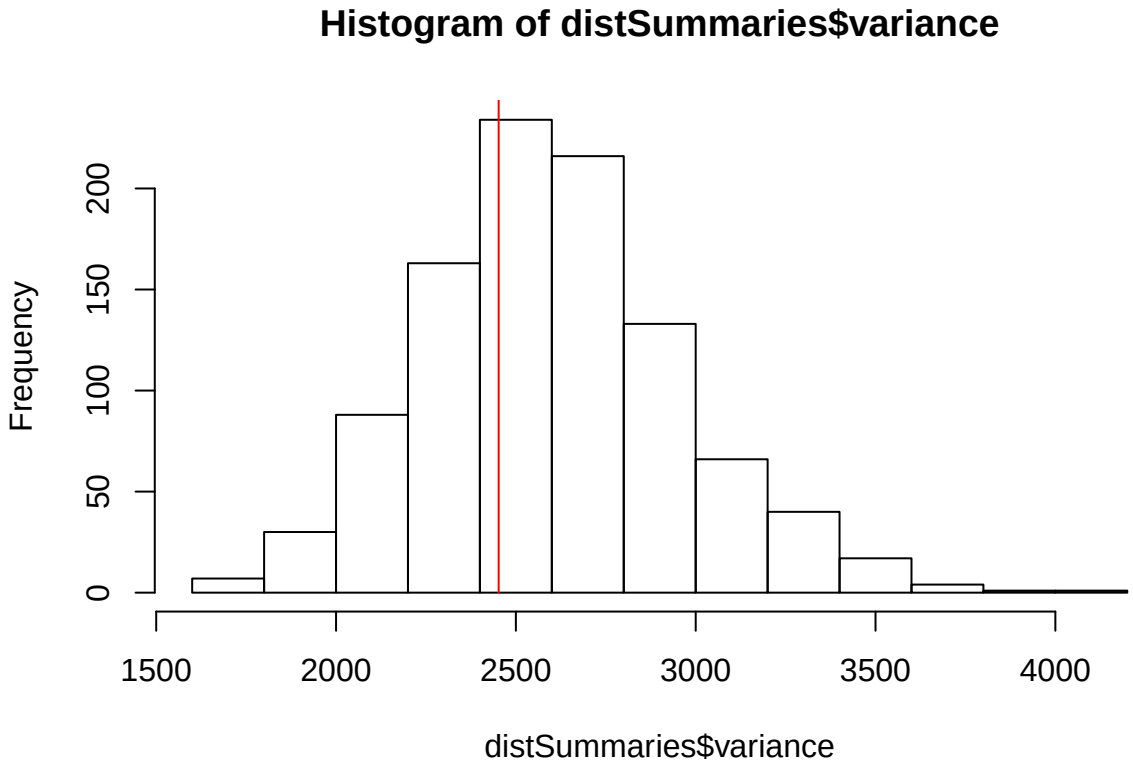
```
## [1] "intercept"     "height"        "group2"        "group3"
## [5] "height.group2" "height.group3"
```

```
parameterNames = c("beta.0", "beta.height", "beta.group2", "beta.group3", "beta.height.group2", "beta.h
```

```r
draws = sample(x = 1:nrow(BayesianFullModel.Posterior.All), size = nSimData, replace = TRUE)

distSummaries = data.frame(mean = matrix(data = NA, nrow = nSimData),
                           variance = matrix(data = NA, nrow = nSimData))

# the following syntax is put into a loop for didactic purposes (a version of apply() would be much fas
i = 1
for (i in 1:nSimData){

  # Grab parameters from iteration in draws
  parameters = BayesianFullModel.Posterior.All[draws[i],]

  # Create Beta vector (which becomes mean of random draw), matching parameters with respective columns
  meanVec = BayesianFullModel.Matrix2 %*% matrix(data = parameters[parameterNames], ncol = 1)

  # draw random Y for all observations
  ppcY = rnorm(n = nrow(BayesianFullModel.Matrix2), mean = meanVec, sd = parameters["sigma.e"])

  # record summaryies of distribution of Y to compare to observed Y
  distSummaries[i,]$mean = mean(ppcY)
  distSummaries[i,]$variance = var(ppcY)

}

# examine how observed sample mean compares to values generated using simulation:
hist(distSummaries$mean, main = "PPC of Sample Mean")
lines(x = c(mean(DietData$WeightLB), mean(DietData$WeightLB)), y = c(0,250), col = 2)
```



```r
# examine how observed sample mean compares to values generated using simulation:
hist(distSummaries$variance)
lines(x = c(var(DietData$WeightLB), var(DietData$WeightLB)), y = c(0,250), col = 2)
```



Results

Our chains look converged, our model fit looks good, now we can look at the results:

```r
# the summary function doesn't print things out well:
summary(BayesianFullModel.Posterior)
```

```
##
## Iterations = 5001:10000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                          Mean        SD  Naive SE Time-series SE
## Rsquare             9.671e-01  0.009067 6.412e-05      7.474e-05
## beta.0              1.708e+02 48.385970 3.421e-01      3.472e-01
## beta.group2        -1.732e+02 67.877682 4.800e-01      4.723e-01
## beta.group3        -1.333e+02 64.021723 4.527e-01      4.471e-01
## beta.height        -3.862e-01  0.757297 5.355e-03      5.431e-03
## beta.height.group2  2.482e+00  1.062998 7.517e-03      7.469e-03
## beta.height.group3  3.577e+00  1.012075 7.156e-03      6.983e-03
## deviance            2.240e+02  7.032384 4.973e-02      6.654e-02
```

```
## dif.group2.group3   3.987e+01 64.521135 4.562e-01      4.605e-01
## sigma.e             1.301e+01  1.932803 1.367e-02      1.679e-02
## sigma2.e            1.730e+02 53.436527 3.779e-01      4.640e-01
## tau.e               6.285e-03  0.001783 1.261e-05      1.512e-05
##
## 2. Quantiles for each variable:
##
##                          2.5%        25%        50%        75%     97.5%
## Rsquare             9.436e-01    0.96342    0.96937  9.733e-01   0.97735
## beta.0              7.393e+01  139.01303  171.02777  2.021e+02 265.44476
## beta.group2        -3.071e+02 -217.85516 -173.00514 -1.285e+02 -40.27762
## beta.group3        -2.596e+02 -175.50529 -132.88512 -9.171e+01  -5.40781
## beta.height        -1.871e+00   -0.87749   -0.38744  1.079e-01   1.12032
## beta.height.group2  3.910e-01    1.78268    2.48167  3.184e+00   4.58555
## beta.height.group3  1.562e+00    2.91743    3.57406  4.242e+00   5.57869
## deviance            2.123e+02  218.93856  223.38443  2.284e+02 239.68384
## dif.group2.group3  -8.691e+01   -2.51107   39.31963  8.169e+01 168.75507
## sigma.e             9.873e+00   11.62916   12.79293  1.413e+01  17.43055
## sigma2.e            9.748e+01  135.23742  163.65905  1.996e+02 303.82395
## tau.e               3.291e-03    0.00501    0.00611  7.394e-03   0.01026
```

```
# inspecting Bayesian Results
BayesianFullModel.Summary = summary(BayesianFullModel.Posterior)
round(BayesianFullModel.Summary$statistics, 3)
```

```
##                         Mean     SD Naive SE Time-series SE
## Rsquare                0.967  0.009    0.000          0.000
## beta.0               170.771 48.386    0.342          0.347
## beta.group2         -173.160 67.878    0.480          0.472
## beta.group3         -133.292 64.022    0.453          0.447
## beta.height           -0.386  0.757    0.005          0.005
## beta.height.group2     2.482  1.063    0.008          0.007
## beta.height.group3     3.577  1.012    0.007          0.007
## deviance             224.017  7.032    0.050          0.067
## dif.group2.group3     39.868 64.521    0.456          0.461
## sigma.e               13.011  1.933    0.014          0.017
## sigma2.e             173.025 53.437    0.378          0.464
## tau.e                  0.006  0.002    0.000          0.000
```

Now, let's compare Bayesian with least squares results:

```
summary(FullModel)
```

```
##
## Call:
## lm(formula = WeightLB ~ HeightIN + factor(DietGroup) + HeightIN:factor(DietGroup),
##     data = DietData)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.6724  -2.7169   0.4958   1.7918  16.4581
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  170.1664    29.2786   5.812 5.42e-06 ***
## HeightIN                      -0.3768     0.4587  -0.822 0.419392
```

```
## factor(DietGroup)2          -172.5639    41.4062  -4.168 0.000345 ***
## factor(DietGroup)3          -132.6675    38.7846  -3.421 0.002241 **
## HeightIN:factor(DietGroup)2    2.4727     0.6486   3.812 0.000846 ***
## HeightIN:factor(DietGroup)3    3.5666     0.6132   5.817 5.36e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.949 on 24 degrees of freedom
## Multiple R-squared:  0.9787, Adjusted R-squared:  0.9742
## F-statistic: 220.3 on 5 and 24 DF,  p-value: < 2.2e-16
```

Often, each parameter is summarized by its HPD (Highest Posterior Density), a value like a confidence interval. To make our life easy, we can use the `HDInterval` package which makes working with the R2jags output easy:

```
hdi(BayesianFullModel.Posterior)
```

```
##         Rsquare    beta.0 beta.group2 beta.group3 beta.height
## lower 0.9497963  72.73014   -306.7909 -260.431515   -1.883774
## upper 0.9787577 263.68190    -40.0764   -6.270331    1.099239
##       beta.height.group2 beta.height.group3 deviance dif.group2.group3
## lower          0.3770387           1.569665 211.2679         -92.06455
## upper          4.5577062           5.584401 237.9753         162.32140
##         sigma.e  sigma2.e       tau.e
## lower  9.576525  85.14509 0.003041983
## upper 16.892455 276.51649 0.009845272
## attr(,"credMass")
## [1] 0.95
```

Reasons for differences: σ^2_e. Note that the values reported above are the mean of σ^2_e, which will be pulled toward the tail of the distribution. We can investigate the mode of σ^2_e, which is more analogous to the MLE (which we get in OLS). Note, the packages below take a while to install. I have commented them out for this example, but if you would like to follow along, please uncomment and provide the time for installation.

```
#
# if (!requireNamespace("BiocManager", quietly = TRUE))
#     install.packages("BiocManager")
# BiocManager::install("genefilter", version = "3.8")
# if (!requireNamespace("modeest")) install.packages("modeest")
# library(modeest)

# mlv(x = BayesianFullModel.Posterior.All[,10], method = "naive")
# mean(BayesianFullModel.Posterior.All[,10])
#
# summary(FullModel)$sigma
```

Look at what happens if we fix the residual variance to that it was in the least squares analysis (now using the R2jags package to save some space)

```
BayesianFullModel.FixedVar = function(){
  # prior distributions:
  #   Note that terms on the left are model parameter names we create here
  beta.0             ~ dnorm(betaMean.0, betaTau.0) # prior for the intercept
  beta.height        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional slope of height
  beta.group2        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference between
  beta.group3        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference between
  beta.height.group2 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 2 (
```

```
  beta.height.group3 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 3 (


  # conditional distribution of the data (uses priors above)
  for (i in 1:N){

    # creating conditional mean to put into model statement
    weight.hat[i] <- beta.0 + beta.height*height[i] + beta.group2*group2[i] + beta.group3*group3[i] + be

    # error values (for R^2)
    error[i] = weight[i]-weight.hat[i]

    # likelihood from model:
    weight[i] ~ dnorm(weight.hat[i], 1/sampleVar)

  }

  # calculate mean difference between groups 2 and 3:
  dif.group2.group3 <- beta.group3 - beta.group2

  # calculate R^2 based on http://www.stat.columbia.edu/~gelman/research/unpublished/bayes_R2.pdf
  variance.pred = sd(weight.hat[])*sd(weight.hat[])
  variance.error = sd(error[])*sd(error[])
  Rsquare = variance.pred/(variance.pred + variance.error)
}

# the parameter names come from the model syntax in the previous step
BayesianFullModel.FixedVar.Parameters = c("beta.0",
                                          "beta.height",
                                          "beta.group2",
                                          "beta.group3",
                                          "beta.height.group2",
                                          "beta.height.group3",
                                          "dif.group2.group3",
                                          "Rsquare")

# add data and N to list object that we will pass to JAGS
BayesianFullModel.FixedVar.Data = list(N = nrow(DietData),
                                       weight = BayesianFullModel.Matrix$weight,
                                       height = BayesianFullModel.Matrix$height,
                                       group2 = BayesianFullModel.Matrix$group2,
                                       group3 = BayesianFullModel.Matrix$group3,
                                       betaMean.0 = betaMean.0,
                                       betaTau.0 = betaTau.0,
                                       sampleVar = summary(FullModel)$sigma^2)

# pick a number: Here is the date I created this syntax
# in order to have reproducable Markov chains, we need to initialize the random number generator and se
# pick a number: Here is the date I created this syntax
BayesianFullModel.FixedVar.Seed = 11022019

#   Note: here, the random number seed cannot be the same per seed or the chains will be the same
RNGname = c("Wichmann-Hill","Marsaglia-Multicarry", "Super-Duper", "Mersenne-Twister")
```

```r
if (RNGname[1] %in% c("Wichmann-Hill", "Marsaglia-Multicarry",
  "Super-Duper", "Mersenne-Twister")) {
  RNGname[1] <- paste("base::", RNGname[1], sep = "")
}

BayesianFullModel.FixedVar.Init.Values <- vector("list", n.chains)
for (i in 1:n.chains) {
  BayesianFullModel.FixedVar.Init.Values[[i]]$.RNG.name <- RNGname[1]
  BayesianFullModel.FixedVar.Init.Values[[i]]$.RNG.seed <- BayesianFullModel.FixedVar.Seed + i
}



BayesianFullModelFixedVar.JAGS = jags(data = BayesianFullModel.FixedVar.Data,
                                      inits = BayesianFullModel.FixedVar.Init.Values,
                                      parameters.to.save = BayesianFullModel.FixedVar.Parameters,
                                      model.file = BayesianFullModel.FixedVar,
                                      n.chains = 4,
                                      n.iter = 50000,
                                      n.burnin = 20000,
                                      n.thin = 1)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 6
##    Total graph size: 256
##
## Initializing model
```

```r
BayesianFullModelFixedVar.JAGS
```

```
## Inference for Bugs model at "/var/folders/sl/n1qxb2m57pqfs1hcfqb8p4nx6fm6zn/T//RtmprovcRx/modeld4426
##  4 chains, each with 50000 iterations (first 20000 discarded)
##  n.sims = 120000 iterations saved
##                     mu.vect sd.vect     2.5%      25%      50%      75%
## Rsquare               0.974   0.003    0.966    0.973    0.975    0.977
## beta.0              170.039  29.343  112.368  150.181  170.113  189.853
## beta.group2        -172.491  41.478 -253.637 -200.511 -172.469 -144.496
## beta.group3        -132.650  38.894 -208.785 -158.904 -132.623 -106.323
## beta.height          -0.375   0.460   -1.272   -0.684   -0.376   -0.064
## beta.height.group2    2.471   0.650    1.194    2.034    2.472    2.910
## beta.height.group3    3.566   0.615    2.364    3.149    3.566    3.981
## dif.group2.group3    39.841  38.704  -35.723   13.656   39.862   65.994
## deviance            209.537   3.473  204.758  206.979  208.888  211.381
##                      97.5%  Rhat  n.eff
## Rsquare              0.978 1.001 120000
## beta.0             227.376 1.001 110000
## beta.group2        -91.228 1.001 120000
## beta.group3        -56.789 1.001 120000
## beta.height          0.528 1.001 110000
## beta.height.group2   3.740 1.001 120000
## beta.height.group3   4.767 1.001 120000
```

```
## dif.group2.group3  115.918 1.001 110000
## deviance           217.994 1.001 120000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 6.0 and DIC = 215.6
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Model Comparison

To show how to compare two models using the DIC (Deviance Information Criterion), we will again revisit the full model, only this time we will model the variance using a log-linear model, making this a location-scale model:

$$\log\left(\frac{1}{\sigma_e^2}\right) = \log(1) - \log\left(\sigma_e^2\right) = -\log\left(\sigma_e^2\right) = \gamma_0$$

Here, γ_0 is a continuous variable that can take be real number. So, we can use a normal prior distribution on γ_0. Note, γ_0 is the value for the intercept of the residual precision $\tau_e = \frac{1}{\sigma_e^2}$ whereas $-\gamma_0$ is the value for the intercept of the variance. As such, this analysis yields a log-normal prior distribution on σ^2, so we are essentially comparing models with different priors.

```
BayesianFullModel = function(){
 # prior distributions:
    #   Note that terms on the left are model parameter names we create here
    beta.0             ~ dnorm(betaMean.0, betaTau.0) # prior for the intercept
    beta.height        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional slope of height
    beta.group2        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwee
    beta.group3        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwee
    beta.height.group2 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 2
    beta.height.group3 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 3
    tau.e              ~ dgamma(alpha.tau.0, beta.tau.0)      #prior for 1/sigma^2_e, residual precisio


    # conditional distribution of the data (uses priors above)
    for (i in 1:N){

      # creating conditional mean to put into model statement
      weight.hat[i] <- beta.0 + beta.height*height[i] + beta.group2*group2[i] + beta.group3*group3[i] +

      # error values (for R^2)
      error[i] = weight[i]-weight.hat[i]

      # likelihood from model:
      weight[i] ~ dnorm(weight.hat[i], tau.e)

    }

    # created values to track throughout the Markov chain
     sigma2.e  <- 1/tau.e                  #create residual variance from 1/precision
     sigma.e   <- sqrt(sigma2.e)           #create residual SD
```

```
    # calculate mean difference between groups 2 and 3:
    dif.group2.group3 <- beta.group3 - beta.group2

    #difRPrec.group2.group3 <- gamma.group3 - gamma.group2

    # calculate R^2 based on http://www.stat.columbia.edu/~gelman/research/unpublished/bayes_R2.pdf
     variance.pred = sd(weight.hat[])*sd(weight.hat[])
     variance.error = sd(error[])*sd(error[])
     Rsquare = variance.pred/(variance.pred + variance.error)
}

BayesianFullModel.Results = jags(data = BayesianFullModel.JAGS.Data, inits = BayesianFullModel.Init.Valu
                                 parameters.to.save = BayesianFullModel.Parameters,
                                 model.file = BayesianFullModel, n.chains = 4, n.iter = 50000, n.burnin
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 7
##    Total graph size: 259
##
## Initializing model
```

```
## Warning in FUN(X[[i]], ...): Failed to set trace monitor for deviance
## Monitor already exists and cannot be duplicated
```

```
BayesianFullModel2 = function(){
 # prior distributions:
    #   Note that terms on the left are model parameter names we create here
    beta.0             ~ dnorm(betaMean.0, betaTau.0) # prior for the intercept
    beta.height        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional slope of height
    beta.group2        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwee
    beta.group3        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwee
    beta.height.group2 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 2
    beta.height.group3 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 3
    gamma.0            ~ dnorm(betaMean.0, betaTau.0) #prior for the log residual precision
    tau.e <- exp(gamma.0)


    # conditional distribution of the data (uses priors above)
    for (i in 1:N){

      # creating conditional mean to put into model statement
      weight.hat[i] <- beta.0 + beta.height*height[i] + beta.group2*group2[i] + beta.group3*group3[i] +

      # error values (for R^2)
      error[i] = weight[i]-weight.hat[i]

      # likelihood from model:
      weight[i] ~ dnorm(weight.hat[i], tau.e)

    }
```

```
    # created values to track throughout the Markov chain
     sigma2.e  <- 1/tau.e                    #create residual variance from 1/precision
     sigma.e   <- sqrt(sigma2.e)             #create residual SD

    # calculate mean difference between groups 2 and 3:
    dif.group2.group3 <- beta.group3 - beta.group2

    #difRPrec.group2.group3 <- gamma.group3 - gamma.group2

    # calculate R^2 based on http://www.stat.columbia.edu/~gelman/research/unpublished/bayes_R2.pdf
     variance.pred = sd(weight.hat[])*sd(weight.hat[])
     variance.error = sd(error[])*sd(error[])
     Rsquare = variance.pred/(variance.pred + variance.error)
}

BayesianFullModel.Results2 = jags(data = BayesianFullModel.JAGS.Data, inits = BayesianFullModel.Init.Va
                                  parameters.to.save = c(BayesianFullModel.Parameters, "gamma.0"),
                                  model.file = BayesianFullModel2, n.chains = 4, n.iter = 50000, n.burni
```

```
## Warning in jags.model(model.file, data = data, inits = init.values,
## n.chains = n.chains, : Unused variable "alpha.tau.0" in data
```

```
## Warning in jags.model(model.file, data = data, inits = init.values,
## n.chains = n.chains, : Unused variable "beta.tau.0" in data
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 7
##    Total graph size: 258
##
## Initializing model
```

```
## Warning in FUN(X[[i]], ...): Failed to set trace monitor for deviance
## Monitor already exists and cannot be duplicated
```

```
BayesianFullModel3 = function(){
 # prior distributions:
    #   Note that terms on the left are model parameter names we create here
    beta.0             ~ dnorm(betaMean.0, betaTau.0) # prior for the intercept
    beta.height        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional slope of height
    beta.group2        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwee
    beta.group3        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwee
    beta.height.group2 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 2
    beta.height.group3 ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of height with group 3
    gamma.0            ~ dnorm(betaMean.0, betaTau.0) #prior for the interaction of residual precision
    gamma.group2        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwe
    gamma.group3        ~ dnorm(betaMean.0, betaTau.0) #prior for the conditional mean difference betwe
    # tau.e <- exp(gamma.0)


    # conditional distribution of the data (uses priors above)
    for (i in 1:N){
```

```
      # creating conditional mean to put into model statement
      weight.hat[i] <- beta.0 + beta.height*height[i] + beta.group2*group2[i] + beta.group3*group3[i] +

      # error values (for R^2)
      error[i] = weight[i]-weight.hat[i]

      varhat[i] = gamma.0 + gamma.group2*group2[i] + gamma.group3*group3[i]
      tau.e[i] <- exp(varhat[i])
      # likelihood from model:
      weight[i] ~ dnorm(weight.hat[i], tau.e[i])

    }

  # calculate mean difference between groups 2 and 3:
    dif.group2.group3 <- beta.group3 - beta.group2

    difRPrec.group2.group3 <- gamma.group3 - gamma.group2

    # calculate R^2 based on http://www.stat.columbia.edu/~gelman/research/unpublished/bayes_R2.pdf
    # variance.pred = sd(weight.hat[])*sd(weight.hat[])
    # variance.error = sd(error[])*sd(error[])
    # Rsquare = variance.pred/(variance.pred + variance.error)
}
BayesianFullModel3.Parameters = c("beta.0",
                                  "beta.height",
                                  "beta.group2",
                                  "beta.group3",
                                  "beta.height.group2",
                                  "beta.height.group3",
                                  "dif.group2.group3",
                                  "difRPrec.group2.group3",
                                  "gamma.0", "gamma.group2", "gamma.group3")

BayesianFullModel.Results3 = jags(data = BayesianFullModel.JAGS.Data, inits = BayesianFullModel.Init.Va
                                  parameters.to.save = BayesianFullModel.FixedVar.Parameters,
                                  model.file = BayesianFullModel2, n.chains = 4, n.iter = 50000, n.burni
```

```
## Warning in jags.model(model.file, data = data, inits = init.values,
## n.chains = n.chains, : Unused variable "alpha.tau.0" in data

## Warning in jags.model(model.file, data = data, inits = init.values,
## n.chains = n.chains, : Unused variable "beta.tau.0" in data
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 7
##    Total graph size: 258
##
## Initializing model
```

```
BayesianFullModel.Results
```

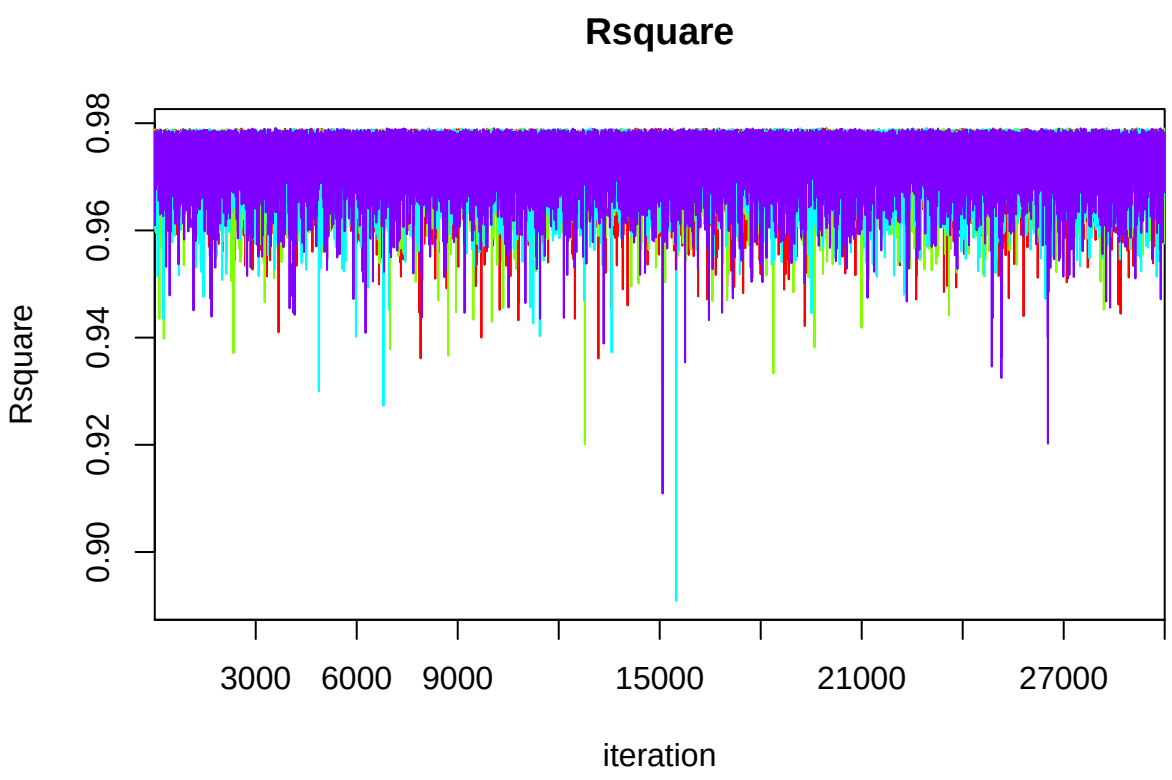
```
## Inference for Bugs model at "/var/folders/sl/n1qxb2m57pqfs1hcfqb8p4nx6fm6zn/T//RtmprovcRx/modeld4427
##  4 chains, each with 50000 iterations (first 20000 discarded)
##  n.sims = 120000 iterations saved
##                      mu.vect sd.vect     2.5%      25%      50%      75%
## Rsquare                0.967   0.009    0.944    0.963    0.969    0.973
## beta.0               170.287  48.242   74.779  138.618  170.231  202.002
## beta.group2         -172.571  68.417 -307.151 -217.666 -172.642 -127.937
## beta.group3         -133.009  64.039 -259.031 -175.175 -133.203  -90.934
## beta.height           -0.379   0.756   -1.867   -0.875   -0.378    0.117
## beta.height.group2     2.473   1.072    0.348    1.771    2.475    3.178
## beta.height.group3     3.572   1.013    1.564    2.909    3.575    4.238
## dif.group2.group3     39.561  64.220  -86.651   -2.403   39.237   81.724
## sigma.e               12.985   1.926    9.873   11.620   12.761   14.100
## sigma2.e             172.313  53.282   97.471  135.024  162.842  198.797
## tau.e                  0.006   0.002    0.003    0.005    0.006    0.007
## deviance             223.906   6.952  212.282  218.914  223.253  228.162
##                       97.5%  Rhat  n.eff
## Rsquare               0.977 1.001 120000
## beta.0              265.233 1.001  53000
## beta.group2         -37.272 1.001  32000
## beta.group3          -6.392 1.001 120000
## beta.height           1.119 1.001  51000
## beta.height.group2    4.584 1.001  32000
## beta.height.group3    5.568 1.001 120000
## dif.group2.group3   166.045 1.001  39000
## sigma.e              17.368 1.001  92000
## sigma2.e            301.659 1.001  92000
## tau.e                 0.010 1.001  92000
## deviance            239.272 1.001  97000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 24.2 and DIC = 248.1
## DIC is an estimate of expected predictive error (lower deviance is better).
```

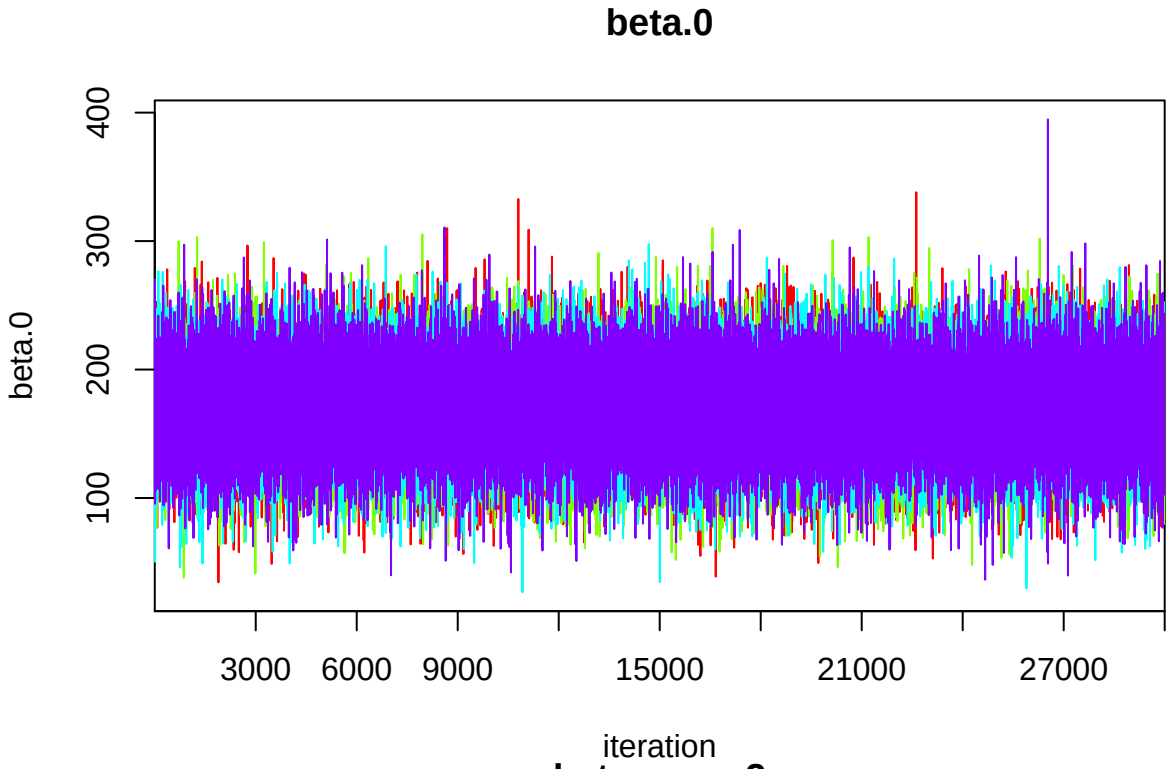
```
BayesianFullModel.Results2
```

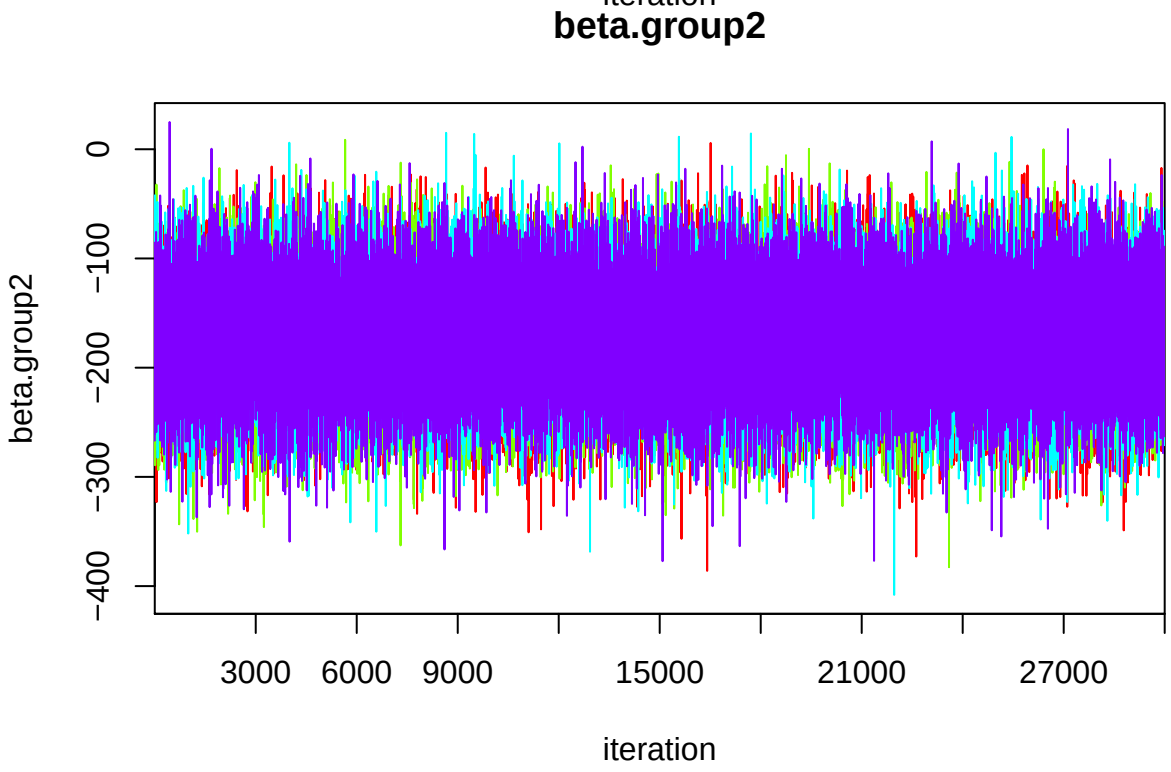
```
## Inference for Bugs model at "/var/folders/sl/n1qxb2m57pqfs1hcfqb8p4nx6fm6zn/T//RtmprovcRx/modeld4423
##  4 chains, each with 50000 iterations (first 20000 discarded)
##  n.sims = 120000 iterations saved
##                      mu.vect sd.vect     2.5%      25%      50%      75%
## Rsquare                0.974   0.004    0.964    0.972    0.975    0.977
## beta.0               170.148  30.595  109.582  149.987  170.160  190.338
## beta.group2         -172.635  43.152 -257.670 -200.965 -172.775 -144.224
## beta.group3         -132.611  40.475 -212.900 -159.262 -132.524 -106.110
## beta.height           -0.377   0.479   -1.321   -0.693   -0.377   -0.061
## beta.height.group2     2.474   0.676    1.143    2.031    2.477    2.917
## beta.height.group3     3.566   0.640    2.305    3.146    3.565    3.985
## dif.group2.group3     40.024  40.522  -40.271   13.335   40.035   66.769
## gamma.0               -4.190   0.296   -4.807   -4.381   -4.176   -3.984
## sigma.e                8.214   1.248    6.206    7.330    8.069    8.941
```

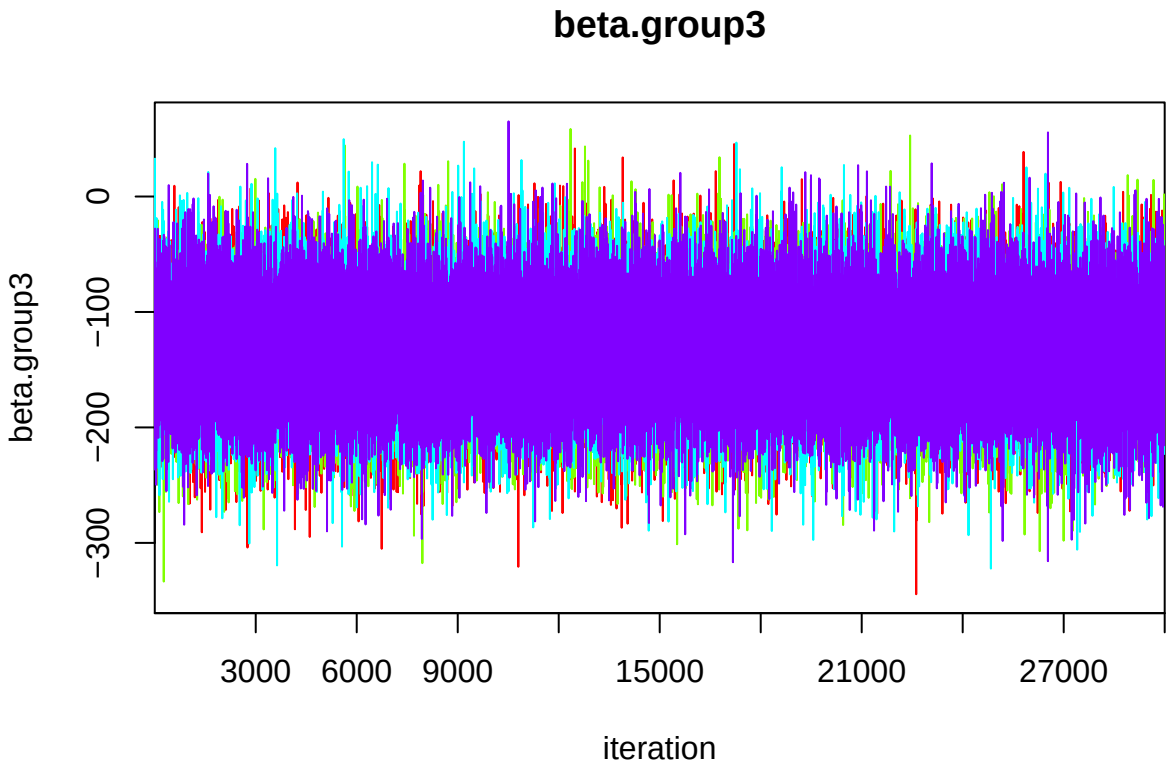
```
## sigma2.e              69.033  21.871   38.519   53.733   65.101   79.944
## tau.e                  0.016   0.005    0.008    0.013    0.015    0.019
## deviance             210.823   4.268  204.769  207.694  210.081  213.150
##                       97.5%  Rhat  n.eff
## Rsquare               0.978 1.001  99000
## beta.0              230.414 1.001 120000
## beta.group2         -87.621 1.001 120000
## beta.group3         -52.758 1.001 120000
## beta.height           0.575 1.001 120000
## beta.height.group2    3.806 1.001 120000
## beta.height.group3    4.836 1.001 120000
## dif.group2.group3   119.693 1.001 120000
## gamma.0              -3.651 1.001 120000
## sigma.e              11.060 1.001 120000
## sigma2.e            122.313 1.001 120000
## tau.e                 0.026 1.001 120000
## deviance            221.141 1.001 120000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 9.1 and DIC = 219.9
## DIC is an estimate of expected predictive error (lower deviance is better).
```

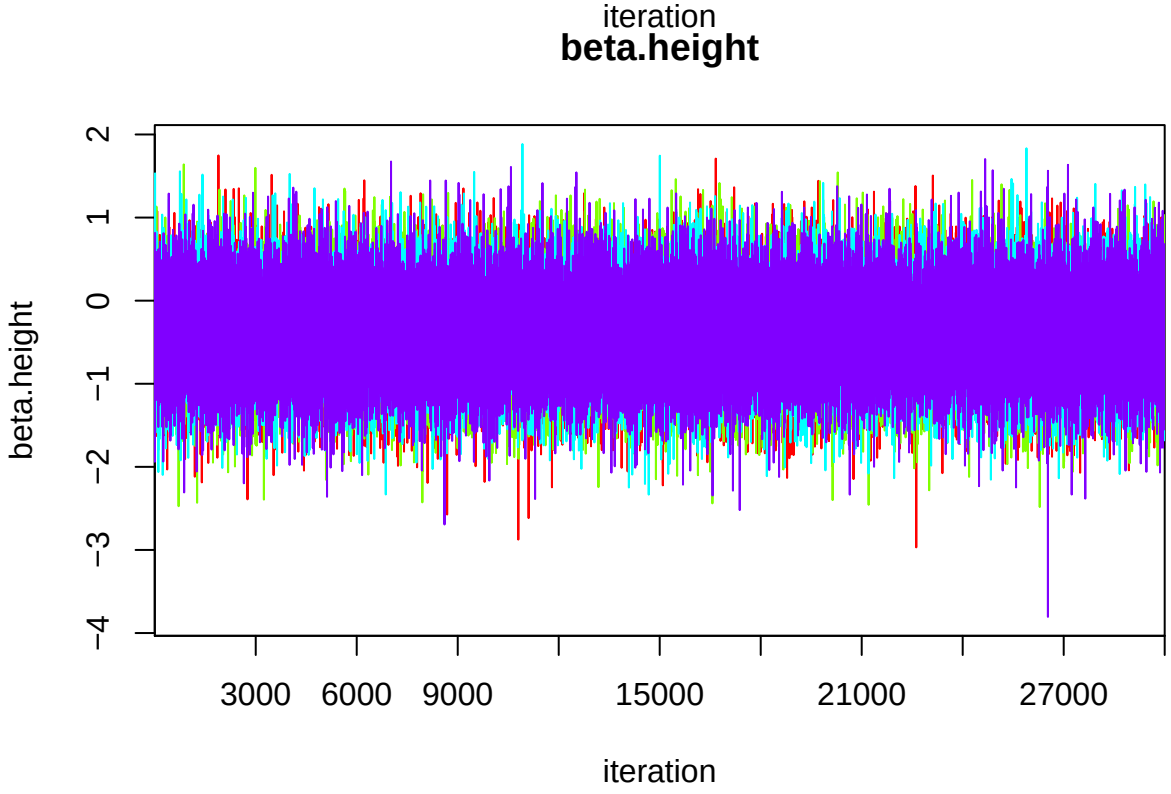
```
traceplot(BayesianFullModel.Results3)
```

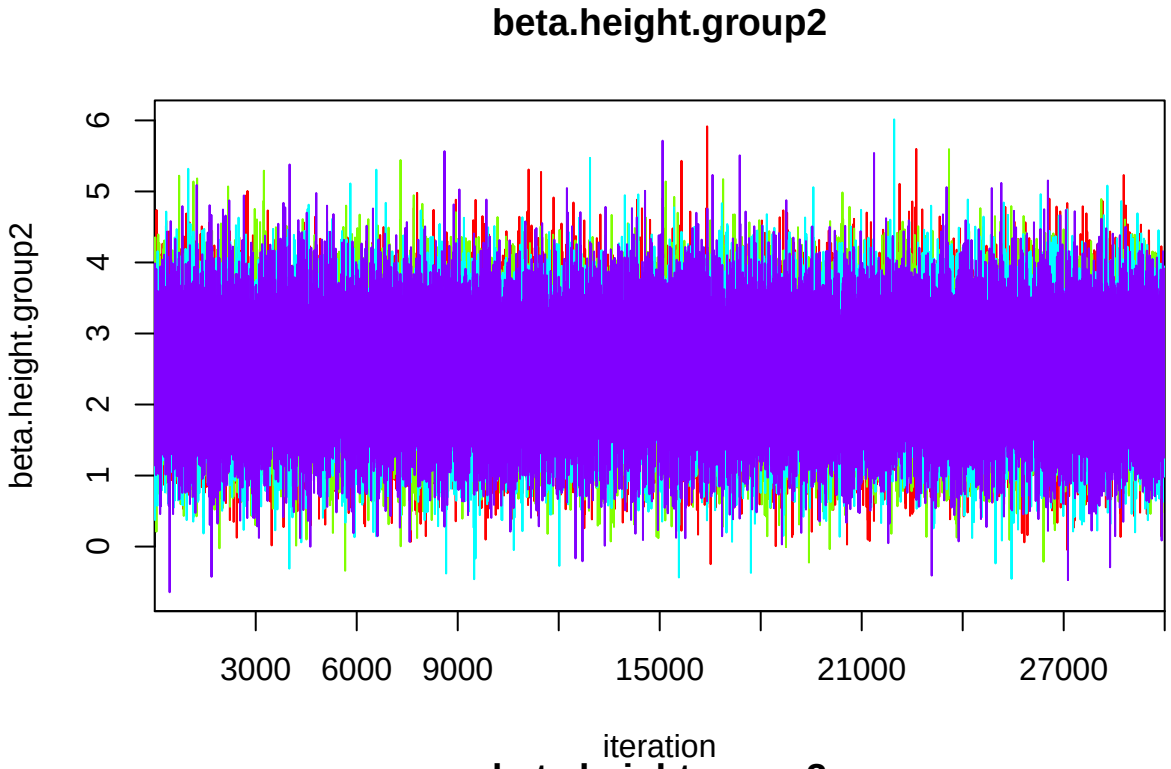


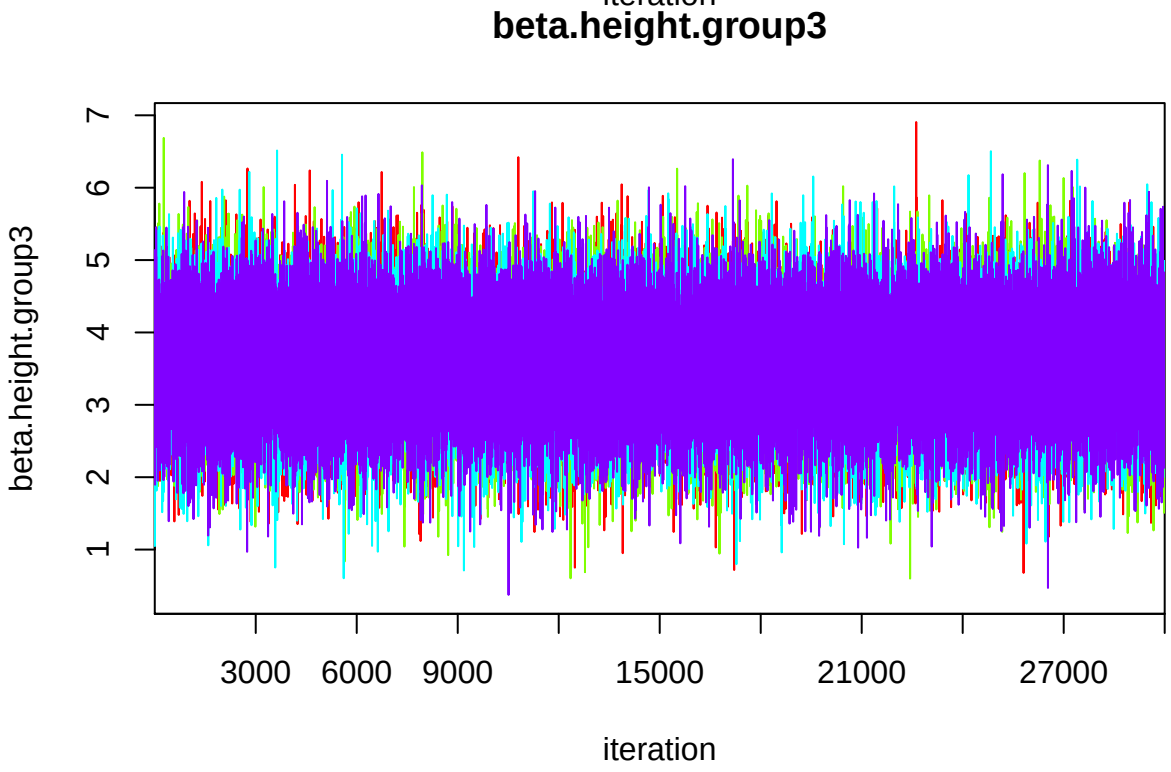

beta.0
beta.0
400
300
200
100
3000
6000
9000
15000
21000
27000
iteration

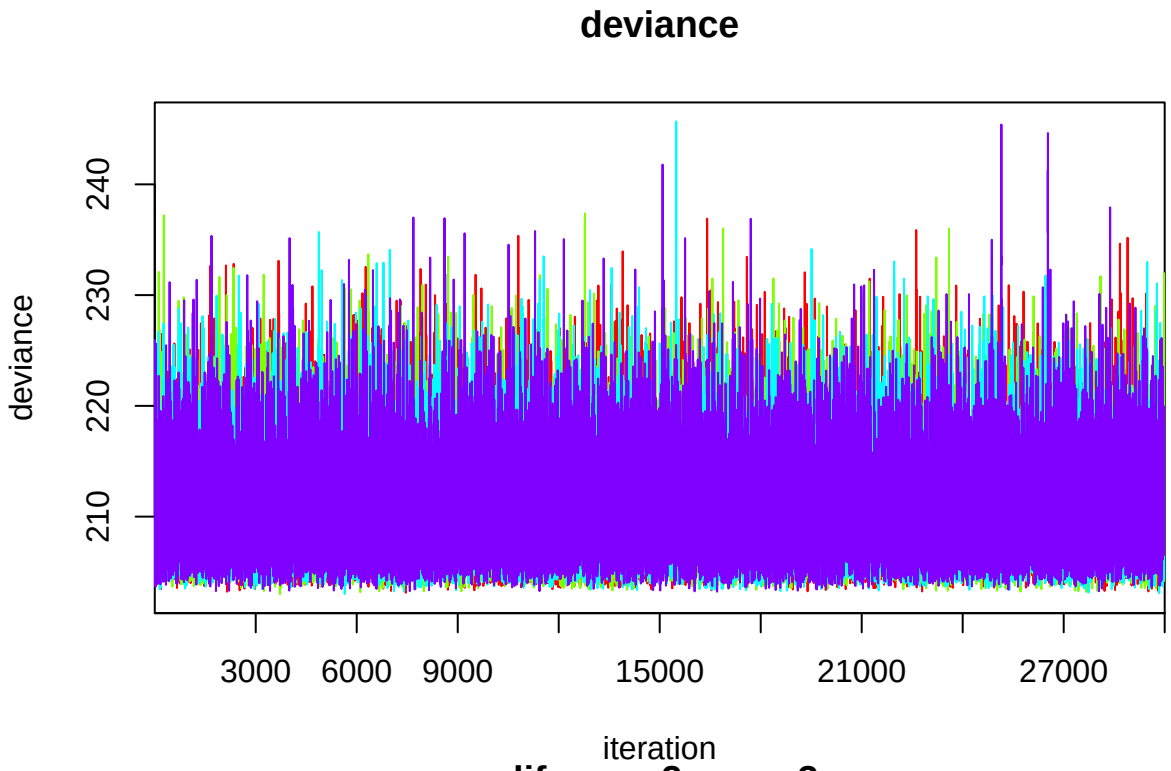

beta.group2
beta.group2
0
−100
−200
−300
−400
3000
6000
9000
15000
21000
27000
iteration

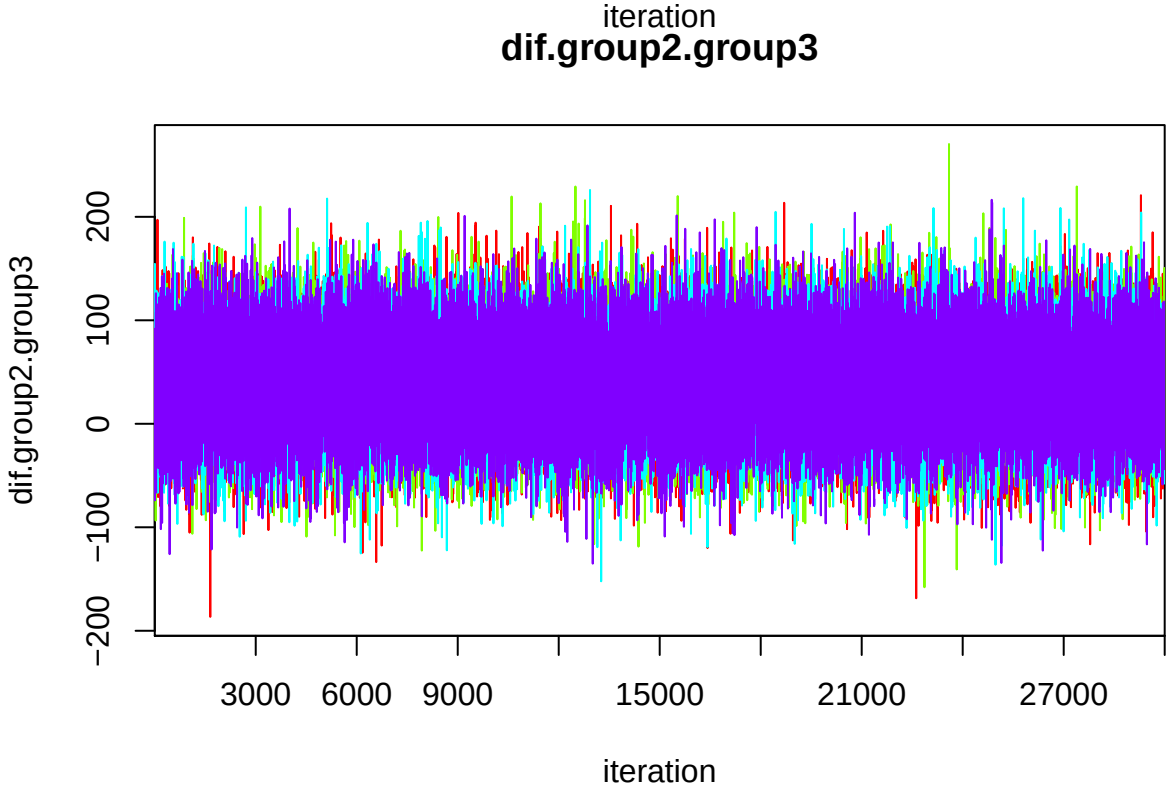

beta.group3
beta.group3
0
−100
−200
−300
3000
6000
9000
15000
21000
27000
iteration


beta.height
beta.height
2
1
0
−1
−2
−3
−4
3000
6000
9000
15000
21000
27000
iteration


beta.height.group2
beta.height.group2
0
1
2
3
4
5
6
3000
6000
9000
15000
21000
27000
iteration


beta.height.group3
beta.height.group3
1
2
3
4
5
6
7
3000
6000
9000
15000
21000
27000
iteration






```
BayesianFullModel.Results3
```

```
## Inference for Bugs model at "/var/folders/sl/n1qxb2m57pqfs1hcfqb8p4nx6fm6zn/T//RtmprovcRx/modeld44293
##  4 chains, each with 50000 iterations (first 20000 discarded)
##  n.sims = 120000 iterations saved
##                     mu.vect sd.vect     2.5%      25%      50%      75%
```

```
## Rsquare               0.974   0.004    0.964    0.972    0.975    0.977
## beta.0              170.063  30.547  109.756  149.972  170.072  190.103
## beta.group2        -172.447  43.213 -257.774 -200.712 -172.401 -143.937
## beta.group3        -132.535  40.432 -212.547 -158.961 -132.575 -106.017
## beta.height          -0.375   0.479   -1.321   -0.690   -0.376   -0.061
## beta.height.group2    2.471   0.677    1.126    2.023    2.471    2.914
## beta.height.group3    3.565   0.639    2.298    3.145    3.566    3.982
## dif.group2.group3    39.912  40.531  -40.295   13.241   39.931   66.571
## deviance            210.783   4.260  204.739  207.665  210.019  213.103
##                      97.5%  Rhat  n.eff
## Rsquare              0.978 1.001  93000
## beta.0             230.395 1.001 120000
## beta.group2        -86.942 1.001  55000
## beta.group3        -52.581 1.001 100000
## beta.height          0.569 1.001 120000
## beta.height.group2   3.810 1.001  58000
## beta.height.group3   4.825 1.001  79000
## dif.group2.group3  120.023 1.001 120000
## deviance           221.030 1.001  45000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 9.1 and DIC = 219.9
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Algorithm Used

Because we changed away from a conjugate prior, we no longer use Gibbs sampling for all parameters. Instead, we now have a "RealSlicer" (a form of Metropolis-Hastings) for γ_0:

```
list.samplers(BayesianFullModel.Results$model)
```

```
## $`glm::Generic`
## [1] "beta.height.group3" "beta.height.group2" "beta.group3"
## [4] "beta.group2"        "beta.height"        "beta.0"
##
## $`bugs::ConjugateGamma`
## [1] "tau.e"
```

```
list.samplers(BayesianFullModel.Results2$model)
```

```
## $`glm::Generic`
## [1] "beta.height.group3" "beta.height.group2" "beta.group3"
## [4] "beta.group2"        "beta.height"        "beta.0"
##
## $`base::RealSlicer`
## [1] "gamma.0"
```

Deviance Information Criterion

The DIC is a Bayesian version of an information criterion used to compare the relative fit of multiple models (see https://en.wikipedia.org/wiki/Deviance_information_criterion). Lower values are better.

The deviance is given by -2 times the likelihood plus a constant (that goes away in model comparisons).

$$D = -2 \log \left(p \left(y \mid \boldsymbol{\theta} \right) \right) + C$$

The DIC is given by the deviance, evaluated at the EAP estimate of all parameters, plus a penalty. The earliest form of the penalty p_D is:

$$p_D = \bar{D} - D \left(\bar{\boldsymbol{\theta}} \right)$$

$$DIC = D \left(\bar{\boldsymbol{\theta}} \right) + 2 p_D$$

Here, the smallest value of the DIC belongs to the model with the log-linear variance prior. So, it is preferred and would be the one interpreted.