Assignment5

Matta Nguyen

2025-04-27

File pigweights.csv contains weekly weights of 48 young pigs for each of 9 consecutive weeks.1. You will build and compare two different varying-coefficient hierarchical normal regression models for the weights, using JAGS and rjags.

#### Question a

On the same set of axes, plot segmented lines, one for each pig, representing the weight versus the week number (1 through 9). Distinguish the lines for different pigs by using different colors or line types. (You should label the axes, but no legend is needed.

pigdata <- read.csv("pigweights.csv")  
  
print("Dimension of the data: ")

## [1] "Dimension of the data: "

print(dim(pigdata))

## [1] 48 10

print("Column Names: ")

## [1] "Column Names: "

print(colnames(pigdata))

## [1] "Pig" "Weight.Week1" "Weight.Week2" "Weight.Week3" "Weight.Week4"  
## [6] "Weight.Week5" "Weight.Week6" "Weight.Week7" "Weight.Week8" "Weight.Week9"

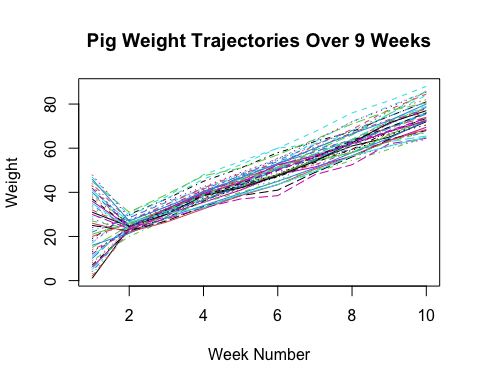
n\_rows <- nrow(pigdata)  
n\_cols <- ncol(pigdata)  
  
print(paste("Number of rows:", n\_rows))

## [1] "Number of rows: 48"

print(paste("Number of columns:", n\_cols))

## [1] "Number of columns: 10"

matplot(1:ncol(pigdata), # x values matching number of columns  
 t(pigdata), # transpose the data  
 type = "l", # 'l' for lines  
 xlab = "Week Number",  
 ylab = "Weight",  
 main = "Pig Weight Trajectories Over 9 Weeks")



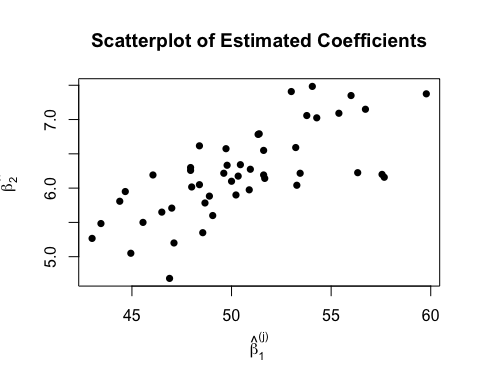
#### Question b

Let ˆβ(j) and ˆβ(j) be the ordinary least squares estimates of β(j) and β(j), estimated for pig j. You may use a function like lm or lsfit in R to compute these estimates. (For this part, the coefficient pairs are estimated completely separately for each pig.)

weight\_data <- pigdata[, 2:ncol(pigdata)]  
  
# Get dimensions  
n\_pigs <- nrow(weight\_data)  
n\_weeks <- ncol(weight\_data)  
weeks <- 1:n\_weeks  
x\_centered <- weeks - mean(weeks)  
  
# Setting up variable for beta1\_hat & beta2\_hat  
beta1\_hat <- numeric(n\_pigs)  
beta2\_hat <- numeric(n\_pigs)  
  
# Fit model  
for(j in 1:n\_pigs) {  
 # Get weights for pig j  
 y <- as.numeric(weight\_data[j,])  
   
 # Fit linear regression  
 fit <- lm(y ~ x\_centered)  
   
 # Store coefficients  
 beta1\_hat[j] <- coef(fit)[1] # intercept  
 beta2\_hat[j] <- coef(fit)[2] # slope  
}

1. Scatter plot

plot(beta1\_hat, beta2\_hat,  
 xlab = expression(hat(beta)[1]^"(j)"),  
 ylab = expression(hat(beta)[2]^"(j)"),  
 main = "Scatterplot of Estimated Coefficients",  
 pch = 16)



1. Give the average (sample mean) of ˆβ(j) and also of ˆβ(j)

#Calculate means  
mean\_beta1 <- mean(beta1\_hat)  
mean\_beta2 <- mean(beta2\_hat)  
cat("\n\nMean of β̂₁:", round(mean\_beta1, 4))

##   
##   
## Mean of β̂₁: 50.4051

cat("\nMean of β̂₂:", round(mean\_beta2, 4))

##   
## Mean of β̂₂: 6.2099

1. [1 pt] Give the sample variance of ˆβ(j) and also of ˆβ(j)

#Calculate variance  
var\_beta1 <- var(beta1\_hat)  
var\_beta2 <- var(beta2\_hat)  
cat("\n\nVariance of β̂₁:", round(var\_beta1, 4))

##   
##   
## Variance of β̂₁: 15.6301

cat("\nVariance of β̂₂:", round(var\_beta2, 4))

##   
## Variance of β̂₂: 0.4066

1. [1 pt] Give the sample correlation between ˆβ(j) and ˆβ(j)

#Calculate sample correlation  
corr\_betas <- cor(beta1\_hat, beta2\_hat)  
cat("\n\nCorrelation between β̂₁ and β̂₂:", round(corr\_betas, 4))

##   
##   
## Correlation between β̂₁ and β̂₂: 0.7127

#### Question c

1. Set up model & run model

model\_string <- "  
model {  
 # Likelihood  
 for (i in 1:N) {  
 for (t in 1:T) {  
 y[i,t] ~ dnorm(mu[i,t], tau.y)  
 mu[i,t] <- beta[i,1] + beta[i,2] \* x[t] # Using centered weeks  
 }  
   
 # Individual-level coefficients with bivariate normal prior  
 beta[i,1:2] ~ dmnorm(mu.beta[], inv.Sigma.beta[,])  
 }  
   
 # Hyperpriors for mean (mu.beta)  
 mu.beta[1] ~ dnorm(0, 0.000001) # precision = 1/1000^2  
 mu.beta[2] ~ dnorm(0, 0.000001)  
   
 # Wishart prior for precision matrix  
 inv.Sigma.beta[1:2,1:2] ~ dwish(inv.Sigma0[,], 2)  
 Sigma.beta[1:2,1:2] <- inverse(inv.Sigma.beta[,])  
   
 # Calculate variance components and correlation  
 sigma.beta1 <- sqrt(Sigma.beta[1,1])  
 sigma.beta2 <- sqrt(Sigma.beta[2,2])  
 rho <- Sigma.beta[1,2]/(sigma.beta1 \* sigma.beta2)  
   
 # Prior for residual variance (inverse gamma via precision)  
 tau.y ~ dgamma(0.0001, 0.0001)  
 sigma2.y <- 1/tau.y  
}"

Run model

library(rjags)

## Loading required package: coda

## Linked to JAGS 4.3.2

## Loaded modules: basemod,bugs

library(coda)  
  
# Set up the data list for JAGS  
Sigma0 <- matrix(c(15, 0, 0, 0.5), 2, 2)  
inv.Sigma0 <- solve(Sigma0)/2   
  
jags\_data <- list(  
 y = as.matrix(weight\_data),  
 x = x\_centered,  
 N = n\_pigs,  
 T = n\_weeks,  
 inv.Sigma0 = inv.Sigma0  
)  
  
# Better initialization function  
init\_values <- function() {  
 # Fit simple linear model to get reasonable starting values  
 time\_data <- rep(x\_centered, each=n\_pigs)  
 weight\_vector <- as.vector(as.matrix(weight\_data))  
 lm\_fit <- lm(weight\_vector ~ time\_data)  
   
 # Use fitted values to initialize  
 list(  
 mu.beta = c(coef(lm\_fit)[1], coef(lm\_fit)[2]),  
 inv.Sigma.beta = solve(Sigma0),   
 tau.y = 1/var(residuals(lm\_fit))   
 )  
}  
  
# Initialize three chains with slightly different starting points  
set.seed(123)   
inits <- list(  
 init\_values(),  
 list(  
 mu.beta = init\_values()$mu.beta \* 1.1, # Slightly different values  
 inv.Sigma.beta = init\_values()$inv.Sigma.beta,  
 tau.y = init\_values()$tau.y \* 1.1  
 ),  
 list(  
 mu.beta = init\_values()$mu.beta \* 0.9,  
 inv.Sigma.beta = init\_values()$inv.Sigma.beta,  
 tau.y = init\_values()$tau.y \* 0.9  
 )  
)  
  
# Compile the model  
jags\_model <- jags.model(textConnection(model\_string),  
 data = jags\_data,  
 inits = inits,  
 n.chains = 3)

## Compiling model graph  
## Resolving undeclared variables  
## Allocating nodes  
## Graph information:  
## Observed stochastic nodes: 432  
## Unobserved stochastic nodes: 52  
## Total graph size: 1475  
##   
## Initializing model

# Longer burn-in  
print("Running burn-in...")

## [1] "Running burn-in..."

update(jags\_model, 10000)  
  
# Monitor parameters with more iterations  
print("Sampling...")

## [1] "Sampling..."

samples <- coda.samples(jags\_model,  
 variable.names = c("mu.beta", "Sigma.beta", "sigma2.y", "rho"),  
 n.iter = 20000,  
 thin = 10) # Added thinning  
  
# Try convergence diagnostics with try()  
print("Checking convergence...")

## [1] "Checking convergence..."

try({  
 gelman <- gelman.diag(samples)  
 print("Gelman-Rubin diagnostics:")  
 print(gelman)  
})

## Error in chol.default(W) : the leading minor of order 3 is not positive

# Check effective sample sizes  
eff\_size <- effectiveSize(samples)  
print("\nEffective sample sizes:")

## [1] "\nEffective sample sizes:"

print(eff\_size)

## Sigma.beta[1,1] Sigma.beta[2,1] Sigma.beta[1,2] Sigma.beta[2,2] mu.beta[1]   
## 5872.947 6000.000 6000.000 6247.966 6541.620   
## mu.beta[2] rho sigma2.y   
## 5881.356 6000.000 6000.000

1. Display the coda summary:

# Summary of results  
print("\nParameter estimates:")

## [1] "\nParameter estimates:"

print(summary(samples))

##   
## Iterations = 10010:30000  
## Thinning interval = 10   
## Number of chains = 3   
## Sample size per chain = 2000   
##   
## 1. Empirical mean and standard deviation for each variable,  
## plus standard error of the mean:  
##   
## Mean SD Naive SE Time-series SE  
## Sigma.beta[1,1] 15.8312 3.48101 0.0449396 0.0454503  
## Sigma.beta[2,1] 1.8438 0.49006 0.0063266 0.0063274  
## Sigma.beta[1,2] 1.8438 0.49006 0.0063266 0.0063274  
## Sigma.beta[2,2] 0.4156 0.09449 0.0012199 0.0011973  
## mu.beta[1] 50.3971 0.57824 0.0074650 0.0071807  
## mu.beta[2] 6.2080 0.09469 0.0012225 0.0012355  
## rho 0.7157 0.07673 0.0009905 0.0009907  
## sigma2.y 1.6056 0.12066 0.0015577 0.0015578  
##   
## 2. Quantiles for each variable:  
##   
## 2.5% 25% 50% 75% 97.5%  
## Sigma.beta[1,1] 10.4270 13.3541 15.3876 17.7293 23.9327  
## Sigma.beta[2,1] 1.0655 1.4962 1.7788 2.1236 2.9918  
## Sigma.beta[1,2] 1.0655 1.4962 1.7788 2.1236 2.9918  
## Sigma.beta[2,2] 0.2663 0.3501 0.4026 0.4679 0.6381  
## mu.beta[1] 49.2669 50.0162 50.3984 50.7777 51.5213  
## mu.beta[2] 6.0212 6.1449 6.2075 6.2714 6.3901  
## rho 0.5446 0.6701 0.7241 0.7705 0.8428  
## sigma2.y 1.3860 1.5216 1.5983 1.6834 1.8593

1. Give an approximate 95% central posterior interval for the correlation parameter ρ, and also produce a graph of its (estimated) posterior density. Does it seem like a good idea to allow ρ to be nonzero

rho\_samples <- do.call(rbind, lapply(samples, function(x) x[,"rho"]))  
  
# Calculate 95% credible interval  
rho\_ci <- quantile(rho\_samples, c(0.025, 0.975))  
print("95% Credible Interval for rho:")

## [1] "95% Credible Interval for rho:"

print(rho\_ci)

## 2.5% 97.5%   
## 0.5445950 0.8428268

Given than 0 is not in the credible interval, it might be a good idea to allow p to be non-zero in this model

1. he population mean regression line represents the expected weight of an “average” pig at week x. Form an approximate 95% central posterior interval for this expected weight at week 1 (x = 1)

extract\_samples <- function(samples) {  
 # Extract mu\_beta samples  
 mu\_beta1 <- do.call(rbind, lapply(samples, function(x) x[,"mu.beta[1]"]))  
 mu\_beta2 <- do.call(rbind, lapply(samples, function(x) x[,"mu.beta[2]"]))  
   
 # Combine into matrix  
 cbind(mu\_beta1, mu\_beta2)  
}  
  
# Get samples  
param\_samples <- extract\_samples(samples)  
  
# Calculate mean weight at week 1  
x <- 1 # Week 1  
x\_centered <- x - mean(weeks)   
  
# Calculate expected weight for each MCMC sample  
expected\_weights <- param\_samples[,1] + param\_samples[,2] \* x\_centered  
  
# Calculate 95% credible interval  
ci <- quantile(expected\_weights, c(0.025, 0.975))  
  
# Print results  
print("95% Credible Interval for expected weight at week 1:")

## [1] "95% Credible Interval for expected weight at week 1:"

print(ci)

## 2.5% 97.5%   
## -151.2424 -150.2058

1. Form an approximate 95% central posterior interval for the population variance of the expected weight at week 1 (x = 1).

# Extract required parameters from MCMC samples  
extract\_variance\_params <- function(samples) {  
 # Extract variance components and correlation  
 sigma\_beta1 <- do.call(rbind, lapply(samples, function(x) x[,"Sigma.beta[1,1]"]))  
 sigma\_beta2 <- do.call(rbind, lapply(samples, function(x) x[,"Sigma.beta[2,2]"]))  
 rho <- do.call(rbind, lapply(samples, function(x) x[,"rho"]))  
   
 # Calculate sigma\_beta1 and sigma\_beta2 as square roots of variances  
 sigma\_beta1 <- sqrt(sigma\_beta1)  
 sigma\_beta2 <- sqrt(sigma\_beta2)  
   
 # Return as matrix  
 cbind(sigma\_beta1, sigma\_beta2, rho)  
}  
  
# Get samples  
variance\_params <- extract\_variance\_params(samples)  
  
# Calculate population variance at week 1  
x <- 1 # Week 1  
x\_centered <- x - mean(weeks) # Center using mean of weeks  
  
# Calculate variance using the formula  
population\_variance <- variance\_params[,1]^2 +   
 2 \* x\_centered \* variance\_params[,3] \* variance\_params[,1] \* variance\_params[,2] +  
 (x\_centered^2) \* variance\_params[,2]^2  
  
# Calculate 95% credible interval for the variance  
variance\_ci <- quantile(population\_variance, c(0.025, 0.975))  
  
# Print results  
print("95% Credible Interval for population variance at week 1:")

## [1] "95% Credible Interval for population variance at week 1:"

print(variance\_ci)

## 2.5% 97.5%   
## -309.2756 -173.5436

1. Simple calculus shows that the population variance of the expected weight of a random pig is minimized at xmin = ¯x − ρ σβ1 /σβ2 Approximate the posterior probability that xmin < 1, i.e., that the minimum occurs before the week 1 time point

# Extract required parameters from MCMC samples  
extract\_params <- function(samples) {  
 # Extract variance components and correlation  
 sigma\_beta1 <- sqrt(do.call(rbind, lapply(samples, function(x) x[,"Sigma.beta[1,1]"])))  
 sigma\_beta2 <- sqrt(do.call(rbind, lapply(samples, function(x) x[,"Sigma.beta[2,2]"])))  
 rho <- do.call(rbind, lapply(samples, function(x) x[,"rho"]))  
   
 # Return as matrix  
 cbind(sigma\_beta1, sigma\_beta2, rho)  
}  
  
# Get samples  
params <- extract\_params(samples)  
  
# Calculate xmin for each MCMC sample  
# xmin = x̄ - ρ \* (σβ1/σβ2)  
xmin <- mean(weeks) - params[,3] \* (params[,1]/params[,2])  
  
# Calculate probability that xmin < 1  
prob\_xmin\_less\_than\_1 <- mean(xmin < 1) # This calculates the proportion of xmin values less than 1  
  
# Print results  
print(paste("Posterior probability that xmin < 1:", round(prob\_xmin\_less\_than\_1, 3)))

## [1] "Posterior probability that xmin < 1: 0.333"

1. The prior probability that xmin < 1 turns out to be approximately 0.205. Approximate the Bayes factor favoring xmin < 1 versus xmin ≥ 1. Then describe the level of data evidence for xmin < 1

# posterior\_prob = mean(xmin < 1)  
posterior\_prob = prob\_xmin\_less\_than\_1   
prior\_prob = 0.205  
  
# Calculate posterior odds  
posterior\_odds = posterior\_prob / (1 - posterior\_prob)  
  
# Calculate prior odds  
prior\_odds = prior\_prob / (1 - prior\_prob)  
  
# Calculate Bayes factor  
bayes\_factor = posterior\_odds / prior\_odds  
  
# Print results  
print(paste("Posterior probability:", round(posterior\_prob, 3)))

## [1] "Posterior probability: 0.333"

print(paste("Posterior odds:", round(posterior\_odds, 3)))

## [1] "Posterior odds: 0.5"

print(paste("Prior odds:", round(prior\_odds, 3)))

## [1] "Prior odds: 0.258"

print(paste("Bayes factor:", round(bayes\_factor, 3)))

## [1] "Bayes factor: 1.939"

# Interpret Bayes factor using common guidelines  
interpret\_bf = function(bf) {  
 if(bf < 1/100) return("Extreme evidence against xmin < 1")  
 else if(bf < 1/30) return("Very strong evidence against xmin < 1")  
 else if(bf < 1/10) return("Strong evidence against xmin < 1")  
 else if(bf < 1/3) return("Moderate evidence against xmin < 1")  
 else if(bf < 3) return("Weak or anecdotal evidence")  
 else if(bf < 10) return("Moderate evidence for xmin < 1")  
 else if(bf < 30) return("Strong evidence for xmin < 1")  
 else if(bf < 100) return("Very strong evidence for xmin < 1")  
 else return("Extreme evidence for xmin < 1")  
}  
  
print(paste("Interpretation:", interpret\_bf(bayes\_factor)))

## [1] "Interpretation: Weak or anecdotal evidence"

viii: Use the rjags function dic.samples to compute the effective number of parameters (“penalty”) and Plummer’s DIC (“Penalized deviance”). Use at least 100,000 iterations

# Compute DIC with 100,000 iterations  
dic\_output\_c <- dic.samples(jags\_model, n.iter=100000)  
  
# Print results  
print("DIC Results:")

## [1] "DIC Results:"

print(dic\_output\_c)

## Mean deviance: 1429   
## penalty 91.07   
## Penalized deviance: 1520

#### Question d

1. Draw DAG for this model

* N(0,1000²) U(0,1000)  
   | |  
   ↓ ↓  
   μβ₁ → β₁⁽ʲ⁾ ←---------------- σβ₁   
   ↘   
   ↘   
   ↘   
   ↓   
   y ← σ²y   
   ↑   
   ↗   
   ↗   
   ↗   
   μβ₂ → β₂⁽ʲ⁾ ←---------------- σβ₂   
   ↑ ↑  
   | |  
   N(0,1000²) U(0,1000)

1. List an appropriate JAGS model. Make sure that there are nodes for σ2β1 , σ2β2 ,and σ2y .

model\_d\_string <- "  
model {  
 # Likelihood  
 for (i in 1:N) {  
 for (t in 1:T) {  
 y[i,t] ~ dnorm(mu[i,t], tau.y)  
 mu[i,t] <- beta[i,1] + beta[i,2] \* weeks[t] # Using weeks directly  
 }  
   
 # Individual-level coefficients with independent normal priors  
 beta[i,1] ~ dnorm(mu.beta1, tau.beta1)  
 beta[i,2] ~ dnorm(mu.beta2, tau.beta2)  
 }  
   
 # Hyperpriors for means  
 mu.beta1 ~ dnorm(0, 0.000001) # precision = 1/1000^2  
 mu.beta2 ~ dnorm(0, 0.000001)  
   
 # Hyperpriors for standard deviations (uniform on SD scale)  
 sigma.beta1 ~ dunif(0, 1000)  
 sigma.beta2 ~ dunif(0, 1000)  
   
 # Convert to precision parameters  
 tau.beta1 <- 1/(sigma.beta1 \* sigma.beta1)  
 tau.beta2 <- 1/(sigma.beta2 \* sigma.beta2)  
   
 # Prior for residual variance (inverse gamma via precision)  
 tau.y ~ dgamma(0.0001, 0.0001)  
   
 # Calculate variance parameters for monitoring  
 sigma2.beta1 <- sigma.beta1 \* sigma.beta1  
 sigma2.beta2 <- sigma.beta2 \* sigma.beta2  
 sigma2.y <- 1/tau.y  
}"  
  
# Set up the data list for JAGS  
jags\_data <- list(  
 y = as.matrix(weight\_data),  
 weeks = 1:n\_weeks, # Using weeks directly  
 N = n\_pigs,  
 T = n\_weeks  
)  
  
# Set up initial values for multiple chains  
init\_values <- function() {  
 list(  
 mu.beta1 = rnorm(1, 0, 10),  
 mu.beta2 = rnorm(1, 0, 10),  
 sigma.beta1 = runif(1, 1, 10),  
 sigma.beta2 = runif(1, 1, 10),  
 tau.y = rgamma(1, 1, 1)  
 )  
}  
  
# Initialize three chains with different starting points  
set.seed(123)  
inits <- list(init\_values(), init\_values(), init\_values())  
  
# Compile the model  
jags\_model <- jags.model(textConnection(model\_d\_string),  
 data = jags\_data,  
 inits = inits,  
 n.chains = 3)

## Compiling model graph  
## Resolving undeclared variables  
## Allocating nodes  
## Graph information:  
## Observed stochastic nodes: 432  
## Unobserved stochastic nodes: 101  
## Total graph size: 1418  
##   
## Initializing model

# Burn-in  
print("Running burn-in...")

## [1] "Running burn-in..."

update(jags\_model, 10000)  
  
# Monitor parameters  
params <- c("mu.beta1", "mu.beta2", "sigma2.beta1", "sigma2.beta2", "sigma2.y")  
print("Sampling...")

## [1] "Sampling..."

samples <- coda.samples(jags\_model,  
 variable.names = params,  
 n.iter = 20000,  
 thin = 10)  
  
# Check convergence  
print("Checking convergence...")

## [1] "Checking convergence..."

gelman <- gelman.diag(samples)  
print("Gelman-Rubin diagnostics:")

## [1] "Gelman-Rubin diagnostics:"

print(gelman)

## Potential scale reduction factors:  
##   
## Point est. Upper C.I.  
## mu.beta1 1 1.00  
## mu.beta2 1 1.00  
## sigma2.beta1 1 1.00  
## sigma2.beta2 1 1.00  
## sigma2.y 1 1.01  
##   
## Multivariate psrf  
##   
## 1

# Check effective sample sizes  
eff\_size <- effectiveSize(samples)  
print("\nEffective sample sizes:")

## [1] "\nEffective sample sizes:"

print(eff\_size)

## mu.beta1 mu.beta2 sigma2.beta1 sigma2.beta2 sigma2.y   
## 6000.000 6000.000 6000.000 6000.000 6016.774

# Summary of results  
print("\nParameter estimates:")

## [1] "\nParameter estimates:"

print(summary(samples))

##   
## Iterations = 11010:31000  
## Thinning interval = 10   
## Number of chains = 3   
## Sample size per chain = 2000   
##   
## 1. Empirical mean and standard deviation for each variable,  
## plus standard error of the mean:  
##   
## Mean SD Naive SE Time-series SE  
## mu.beta1 19.3559 0.41360 0.005340 0.005340  
## mu.beta2 6.2114 0.09514 0.001228 0.001228  
## sigma2.beta1 7.4456 1.80637 0.023320 0.023322  
## sigma2.beta2 0.4042 0.09370 0.001210 0.001210  
## sigma2.y 1.6042 0.12282 0.001586 0.001581  
##   
## 2. Quantiles for each variable:  
##   
## 2.5% 25% 50% 75% 97.5%  
## mu.beta1 18.5439 19.078 19.3582 19.6282 20.1762  
## mu.beta2 6.0246 6.147 6.2126 6.2736 6.4008  
## sigma2.beta1 4.6115 6.180 7.1652 8.4681 11.6596  
## sigma2.beta2 0.2606 0.338 0.3914 0.4558 0.6325  
## sigma2.y 1.3775 1.519 1.5985 1.6837 1.8579

1. Recall the expected weight at week 1 (x = 1) of an “average” pig, as considered in the previous analysis. Form an approximate 95% central posterior interval for this expected weight, and compare it with the result from the previous model.

# Extract required parameters from MCMC samples for new model  
extract\_samples <- function(samples) {  
 # Extract mu\_beta samples  
 mu\_beta1 <- do.call(rbind, lapply(samples, function(x) x[,"mu.beta1"]))  
 mu\_beta2 <- do.call(rbind, lapply(samples, function(x) x[,"mu.beta2"]))  
   
 # Combine into matrix  
 cbind(mu\_beta1, mu\_beta2)  
}  
  
# Get samples  
param\_samples <- extract\_samples(samples)  
  
# Calculate mean weight at week 1  
x <- 1 # Week 1  
expected\_weights <- param\_samples[,1] + param\_samples[,2] \* x  
  
# Calculate 95% credible interval for new model  
ci\_new <- quantile(expected\_weights, c(0.025, 0.975))  
  
# Print results  
print("95% Credible Interval for expected weight at week 1 (new model):")

## [1] "95% Credible Interval for expected weight at week 1 (new model):"

print(ci\_new)

## 2.5% 97.5%   
## 38.27419 39.40154

# Print previous model's results for comparison  
print("\nPrevious model's 95% Credible Interval:")

## [1] "\nPrevious model's 95% Credible Interval:"

print(ci) # from the previous analysis

## 2.5% 97.5%   
## -151.2424 -150.2058

# Compare intervals  
print("\nComparison:")

## [1] "\nComparison:"

print(paste("New model interval width:", round(diff(ci\_new), 3)))

## [1] "New model interval width: 1.127"

print(paste("Previous model interval width:", round(diff(ci), 3)))

## [1] "Previous model interval width: 1.037"

jags\_model <- jags.model(textConnection(model\_d\_string),  
 data = jags\_data,  
 inits = inits,  
 n.chains = 3)

## Compiling model graph  
## Resolving undeclared variables  
## Allocating nodes  
## Graph information:  
## Observed stochastic nodes: 432  
## Unobserved stochastic nodes: 101  
## Total graph size: 1418  
##   
## Initializing model

# Burn-in  
update(jags\_model, 10000)  
  
# Compute DIC with 100,000 iterations  
dic\_output\_b <- dic.samples(jags\_model,   
 type = "pD", # specify the type of DIC  
 n.iter = 100000)  
  
# Print results  
print("DIC Results:")

## [1] "DIC Results:"

print(dic\_output\_b)

## Mean deviance: 1430   
## penalty 90.34   
## Penalized deviance: 1521

1. Compare the two DIC results:

# Print results  
print("DIC Results at example c:")

## [1] "DIC Results at example c:"

print(dic\_output\_c)

## Mean deviance: 1429   
## penalty 91.07   
## Penalized deviance: 1520

print("DIC Results at example d:")

## [1] "DIC Results at example d:"

print(dic\_output\_b)

## Mean deviance: 1430   
## penalty 90.34   
## Penalized deviance: 1521

DIC values are almost identical, therefore we might prefer the simpler model since it’s achiving the same result with simplier structure