HW7

Jaime Romero Florez

2023-12-16

Question 1

```r
file <- "http://www.stat.duke.edu/~pdh10/FCBS/Exercises/interexp.dat"
Data <- read.table(file=file, header=T)
rm(file)
```

a)

```r
Data_complete <- na.omit(Data)

# run Gibbs
# Set up priors
ybar <- colMeans(Data_complete) # sample means
Lambda0 <- cov(Data_complete) # sample covariance
nu.0 <- 2 + 2 #p + 2
S.0 <- Lambda0
p <- dim(Data_complete)[2]
n <- dim(Data_complete)[1]
Cov.y <- cov(Data_complete)


# Gibbs sampling
S <- 10000
mu.0 <- ybar;  Sigma <- S.0;  # Initial values

# Calculations that will be used repeatedly
Lambda0.inv <- solve(Lambda0)
Lam.inv.mu.0 <- Lambda0.inv %*% mu.0
nu.n <- nu.0 + n

# Now generate the Markov chain!  mu and Sigma
mu.chain <- matrix(NA, S, p)
Sigma.chain <- matrix(NA, S, p^2)

for(s in 1:S)
{
 n.Sigma.inv <- n * solve(Sigma)
 Lambda.n <- solve( Lambda0.inv + n.Sigma.inv )
 mu.n <- Lambda.n %*% (Lam.inv.mu.0 + n.Sigma.inv %*% ybar)
 mu <- rmvnorm(1, mu.n, Lambda.n)[1,]
 S.n <- S.0 + (n-1)*Cov.y + n * (ybar-mu) %*% t(ybar-mu)
 Sigma <- solve( rWishart(1, nu.n, solve(S.n))[,,1] )
 mu.chain[s,] <- mu
 Sigma.chain[s,] <- Sigma
}


# Posterior expectation
post.exp1 <- colMeans(mu.chain)
post.exp1
```

```
## [1] 24.14103 24.76129
```

```
# 90% Posterior CI for mu_b - mu_a
quant.1 <- quantile((mu.chain[,2]-mu.chain[,1]), probs = c(0.5, 0.95))
quant.1
```

```
##       50%       95%
## 0.6190643 1.2430822
```

b)

```
# calculating empirical estimates
mu_a <- mean(Data$yA, na.rm = T)
sd_a <- sd(Data$yA, na.rm = T)
mu_b <- mean(Data$yB, na.rm = T)
sd_b <- sd(Data$yB, na.rm = T)
rho <- cor(Data_complete$yA, Data_complete$yB)

# now impute values as "real data"

for (b in 27:41) {
  value <- mu_b + rho * (sd_b / sd_a) * (Data$yA[b] - mu_a)
  Data$yB[b] <- value
}

for (a in 42:58) {
  value <- mu_a + rho * (sd_a / sd_b) * (Data$yB[a] - mu_b)
  Data$yA[a] <- value
}

# now I have a complete dataset!

# now, run Gibbs

# Set up priors
ybar <- colMeans(Data) # sample means
Lambda0 <- cov(Data) # sample covariance
```

```r
nu.0 <- 2 + 2 #p + 2
S.0 <- Lambda0
p <- dim(Data)[2]
n <- dim(Data)[1]
Cov.y <- cov(Data)

# Gibbs sampling
S <- 5000
mu.0 <- ybar;  Sigma <- S.0;  # Initial values

# Calculations that will be used repeatedly
Lambda0.inv <- solve(Lambda0)
Lam.inv.mu.0 <- Lambda0.inv %*% mu.0
nu.n <- nu.0 + n

# Now generate the Markov chain!  mu and Sigma
mu.chain <- matrix(NA, S, p)
Sigma.chain <- matrix(NA, S, p^2)

for(s in 1:S)
{
 n.Sigma.inv <- n * solve(Sigma)
 Lambda.n <- solve( Lambda0.inv + n.Sigma.inv )
 mu.n <- Lambda.n %*% (Lam.inv.mu.0 + n.Sigma.inv %*% ybar)
 mu <- rmvnorm(1, mu.n, Lambda.n)[1,]
 S.n <- S.0 + (n-1)*Cov.y + n * (ybar-mu) %*% t(ybar-mu)
 Sigma <- solve( rWishart(1, nu.n, solve(S.n))[,,1] )
 mu.chain[s,] <- mu
 Sigma.chain[s,] <- Sigma
}


# Posterior expectation
post.exp2 <- colMeans(mu.chain)
post.exp2
```

```
## [1] 24.20819 24.82335
```

```r
# 90% Posterior CI for mu_b - mu_a
quant.2 <- quantile((mu.chain[,2]-mu.chain[,1]), probs = c(0.5, 0.95))
quant.2
```

```
##       50%       95%
## 0.6151087 0.9244181
```

c

Now, with imputation done correctly

```r
# Input:
```

```
# Data matrix y, with missing values indicated by NA

# Parameters for semiconjugate prior given multivariate normal
#  sampling model:  mu.0,  Lam.0,  S.0,  nu.0.

# Starting value for Gibbs sampler:  mu and Sigma

# Desired chain length S


# Output: An (S x p) matrix mu.chain, and an (S x p^2) matrix
#  Sigma.chain, containing posterior simulations


file <- "http://www.stat.duke.edu/~pdh10/FCBS/Exercises/interexp.dat"
y <- as.matrix(read.table(file=file, header=T))
rm(file)
# Gibbs code:

# Set up priors
ybar <- colMeans(y, na.rm = T) # sample means
Lam.0 <- cov(Data_complete) # sample covariance
nu.0 <- 2 + 2 #p + 2
S.0 <- Lambda0
Cov.y <- cov(Data_complete)

n <- dim(y)[1];  p <- dim(y)[2];
I <- !is.na(y) # inclusion matrix (1 for observed, 0 for missing)

y.hat <- y

Lam0.inv <- solve(Lam.0);

mu.chain <- matrix(NA, S, p)

Sigma.chain <- matrix(NA, S, p^2)

for(s in 1:S)
{
 ###
 # Impute missing data first
 ###
 for(i in 1:n){
  b <- ( I[i,]==0 );  if(any(b)){
  a <- ( I[i,]==1 )
  inv.Sa <- solve(Sigma[a,a])
  Sigma.j <- Sigma[b,b] - Sigma[b,a] %*% inv.Sa %*% Sigma[a,b]
  mu.j <- mu[b] + Sigma[b,a] %*% inv.Sa %*% (t(y.hat[i,a])-mu[a])
  y.hat[i,b] <- rmvnorm(1, mu.j, Sigma.j) } }
 ###
 # Update mu and Sigma in the usual way, using y.hat as 'data'
 ###
 # First update mu
 ###
 ybar <- apply(y.hat, 2, mean)
 Cov.y <- cov(y.hat)
 Sig.inv <- solve(Sigma)
 Lam.n <- solve( Lam0.inv + n * Sig.inv )
 mu.n <- Lam.n %*% (Lam0.inv %*% mu.0 + n * Sig.inv %*% ybar)
 mu <- rmvnorm(1, mu.n, Lam.n)[1,]
 ###
 # Now update Sigma
 ###
 Sig.n <- S.0 + (n-1) * Cov.y + n * (ybar-mu) %*% t(ybar-mu)
 Sigma <- solve( rWishart(1, nu.0+n, solve(Sig.n))[,,1] )
 ###
 # Save results
 ###
 mu.chain[s,] <- mu
 Sigma.chain[s,] <- Sigma
}

# Posterior expectation
post.exp3 <- colMeans(mu.chain)
post.exp3
```

```
## [1] 24.20869 24.81232
```

```
# 90% Posterior CI for mu_b - mu_a
quant.3 <- quantile((mu.chain[,2]-mu.chain[,1]), probs = c(0.5, 0.95))
quant.3
```

```
##       50%       95%
## 0.6060942 1.1192162
```

d

```
# Create a data frame for posterior expectations
posterior_expectations <- data.frame(
  Procedures = c("Procedure 1", "Procedure 2", "Procedure 3"),
  muA = c(post.exp1[1], post.exp2[1], post.exp3[1]),
  muB = c(post.exp1[2], post.exp2[2], post.exp3[2])
)

print(posterior_expectations)
```

```
##    Procedures      muA      muB
## 1 Procedure 1 24.14103 24.76129
## 2 Procedure 2 24.20819 24.82335
## 3 Procedure 3 24.20869 24.81232
```

```r
# Create a data frame for confidence intervals
confidence_intervals <- data.frame(
  Procedures = c("Procedure 1 (muB - muA)", "Procedure 2 (muB - muA)",
                 "Procedure 3 (muB - muA)"),
  Lower_Bound = c(quant.1[1], quant.2[1], quant.3[1]),
  Upper_Bound = c(quant.1[2], quant.2[2], quant.3[2])
)

print(confidence_intervals)
```

```
##                Procedures Lower_Bound Upper_Bound
## 1 Procedure 1 (muB - muA)   0.6190643   1.2430822
## 2 Procedure 2 (muB - muA)   0.6151087   0.9244181
## 3 Procedure 3 (muB - muA)   0.6060942   1.1192162
```

```r
rm(list = ls())
```

Comment: For all three procedures, the mean values remained fairly similar, as it can be seen above. It is interesting to see that the highest means are from (c), perhaps as we were simulating the missing values within the Gibbs sampler, they could attain higher values.

With respect to the difference between the variables, they remained positive, indicating that with 95% probability $\mu_B > \mu_A$. This was consistent in all cases, with procedure 1 and 3 (a and c) having the highest upper bounds for the difference at 1.23 and 1.16 approximately. All had lower values of approximately 0.62.

Question 2

```r
file <- "http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/pdensity.dat"
Data <- read.table(file=file, header=T)
rm(file)
```

a)

```r
models <- lapply(split(Data, Data$plot), function(subset) {
  lm(yield ~ density + I(density^2), data = subset)
})

# Display the summary of each model within groups
summaries <- lapply(models, summary)
summaries
```

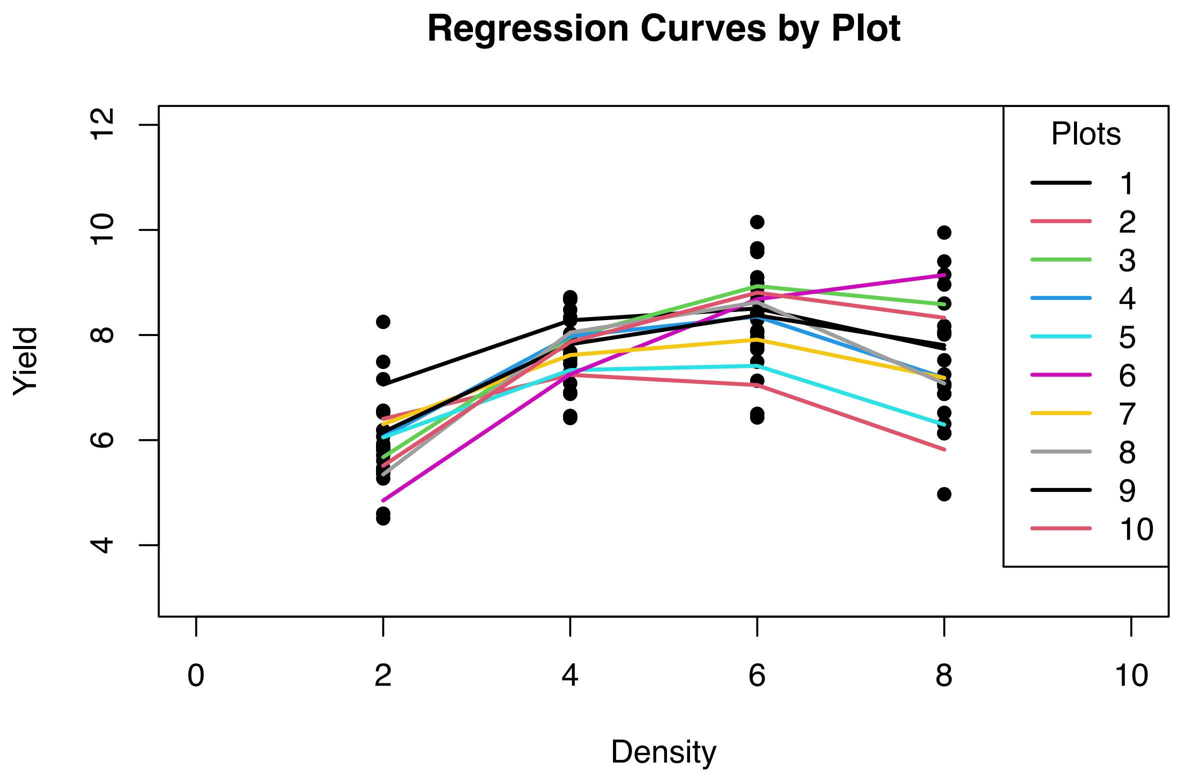
```
## $`1`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##      1      2      3      4      5      6      7      8
##  1.193 -1.247  0.411 -0.249 -0.546  0.384 -1.608  1.662
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.8400     2.6414   1.832    0.126
## density        1.3573     1.2049   1.126    0.311
## I(density^2)  -0.1244     0.1186  -1.049    0.342
##
## Residual standard error: 1.342 on 5 degrees of freedom
## Multiple R-squared:  0.217,  Adjusted R-squared:  -0.09617
## F-statistic: 0.6929 on 2 and 5 DF,  p-value: 0.5425
##
##
## $`2`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##       9      10      11      12      13      14      15      16
## -0.9343  1.0858  0.3678 -0.8223  1.0022 -0.5478 -0.8507  0.6993
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.53375    2.04199   2.220   0.0771 .
## density       1.19338    0.93143   1.281   0.2563
## I(density^2) -0.12906    0.09169  -1.408   0.2183
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.037 on 5 degrees of freedom
## Multiple R-squared:  0.3181, Adjusted R-squared:  0.04541
## F-statistic: 1.166 on 2 and 5 DF,  p-value: 0.3839
##
##
## $`3`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##      17      18      19      20      21      22      23      24
## -0.2365  0.2035  0.3795 -0.2805 -0.2695  0.1705  1.3665 -1.3335
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.07750    1.77543   1.170   0.2947
## density       2.12825    0.80985   2.628   0.0466 *
## I(density^2) -0.16438    0.07972  -2.062   0.0942 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9019 on 5 degrees of freedom
## Multiple R-squared:  0.7596, Adjusted R-squared:  0.6634
## F-statistic: 7.897 on 2 and 5 DF,  p-value: 0.02835
##
##
## $`4`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##       25       26       27       28       29       30       31       32
##  0.12775 -0.13225  0.69175 -0.67825  0.04825 -0.06175 -0.87275  0.87725
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.60375    1.39465   1.867   0.1209
## density       2.11488    0.63616   3.324   0.0209 *
## I(density^2) -0.19281    0.06262  -3.079   0.0275 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7085 on 5 degrees of freedom
## Multiple R-squared:  0.7103, Adjusted R-squared:  0.5944
## F-statistic:  6.13 on 2 and 5 DF,  p-value: 0.04517
##
##
## $`5`
##
```

```
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##     33     34     35     36     37     38     39     40
##  0.019 -0.441  1.148  0.118 -0.983 -0.283 -0.164  0.586
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.57000    1.51033   2.364   0.0645 .
## density       1.54050    0.68892   2.236   0.0756 .
## I(density^2) -0.15000    0.06782  -2.212   0.0779 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7672 on 5 degrees of freedom
## Multiple R-squared:  0.5002, Adjusted R-squared:  0.3003
## F-statistic: 2.502 on 2 and 5 DF,  p-value: 0.1766
##
##
## $`6`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##       41       42       43       44       45       46       47       48
## -0.24925  0.42075 -0.79225  0.27775 -0.95275  1.46725 -0.18075  0.00925
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.47375    1.76887   0.833   0.4427
## density       1.93088    0.80685   2.393   0.0621 .
## I(density^2) -0.12156    0.07942  -1.531   0.1864
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8986 on 5 degrees of freedom
## Multiple R-squared:  0.847,  Adjusted R-squared:  0.7858
## F-statistic: 13.84 on 2 and 5 DF,  p-value: 0.009152
##
##
## $`7`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##      49      50      51      52      53      54      55      56
## -0.4423  0.8578 -0.7082 -0.5382  1.6682 -0.4218 -0.1127 -0.3027
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.96375    1.92631   2.058   0.0947 .
## density       1.42488    0.87867   1.622   0.1658
## I(density^2) -0.12781    0.08649  -1.478   0.1995
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9786 on 5 degrees of freedom
## Multiple R-squared:  0.3814, Adjusted R-squared:  0.134
## F-statistic: 1.542 on 2 and 5 DF,  p-value: 0.3009
##
##
## $`8`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##       57       58       59       60       61       62       63       64
##  0.01375  0.05375  0.23375 -0.43625 -0.82375  1.02625 -0.04875 -0.01875
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.52375    1.23954   0.423  0.69019
## density       2.94188    0.56540   5.203  0.00346 **
## I(density^2) -0.26531    0.05566  -4.767  0.00503 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6297 on 5 degrees of freedom
## Multiple R-squared:  0.8616, Adjusted R-squared:  0.8063
## F-statistic: 15.57 on 2 and 5 DF,  p-value: 0.007122
##
##
## $`9`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##      65      66      67      68      69      70      71      72
## -0.4435  0.3665  0.0955  0.1355 -0.2955  0.0645  0.3635 -0.2865
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.36250    0.71746   4.687  0.00540 **
## density       1.67550    0.32726   5.120  0.00371 **
## I(density^2) -0.14000    0.03221  -4.346  0.00739 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3645 on 5 degrees of freedom
## Multiple R-squared:  0.893,  Adjusted R-squared:  0.8502
## F-statistic: 20.87 on 2 and 5 DF,  p-value: 0.003743
##
##
```

```
## $`10`
##
## Call:
## lm(formula = yield ~ density + I(density^2), data = subset)
##
## Residuals:
##       73       74       75       76       77       78       79       80
##  1.04775 -1.00225  0.85175 -0.98825  0.18325 -0.04675 -0.31775  0.27225
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.73875    1.76404   0.986   0.3696
## density       2.24112    0.80465   2.785   0.0387 *
## I(density^2) -0.17719    0.07921  -2.237   0.0755 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8961 on 5 degrees of freedom
## Multiple R-squared:  0.7616, Adjusted R-squared:  0.6662
## F-statistic: 7.986 on 2 and 5 DF,  p-value: 0.02775
```

i)

```r
# Create an empty plot
plot(1, type = "n", xlim = c(0, 10), ylim = c(3, 12), xlab = "Density", ylab = "Yield", main = "Regressi

# Plotting actual data points
points(Data$density, Data$yield, col = "black", pch = 16)

# Plotting fitted regression curves for each plot
for (i in 1:length(models)) {
  lines(Data$density[Data$plot == names(models)[i]],
        predict(models[[i]], data.frame(density = Data$density[Data$plot == names(models)[i]])),
        col = i, lwd = 2)
}

# Adding a legend for plots
legend("topright", legend = names(models), col = 1:length(models), lwd = 2, title = "Plots")
```



ii)

```
# Extracting OLS coefficients from the fitted models
coefficients_matrix <- sapply(models, coef)

# Transposing the coefficients matrix for easier manipulation
coefficients_matrix <- t(coefficients_matrix)

# Calculating mean vector and covariance matrix of coefficients
mean_coefs <- colMeans(coefficients_matrix)
cov_coefs <- cov(coefficients_matrix)


# Extracting residuals from the models
residuals_list <- lapply(models, function(model) residuals(model))

# Combine residuals from all groups
all_residuals <- unlist(residuals_list)

# Estimated overall variance
sigma_squared_hat <- var(all_residuals)

mean_coefs
```

```
##  (Intercept)      density I(density^2)
##      2.86875      1.85485     -0.15925
```

```
cov_coefs
```

```
##               (Intercept)     density I(density^2)
## (Intercept)   2.00120764 -0.69321312  0.044309549
## density      -0.69321312  0.27555421 -0.020742679
## I(density^2)  0.04430955 -0.02074268  0.001968451
```

```
sigma_squared_hat
```

```
## [1] 0.4986139
```

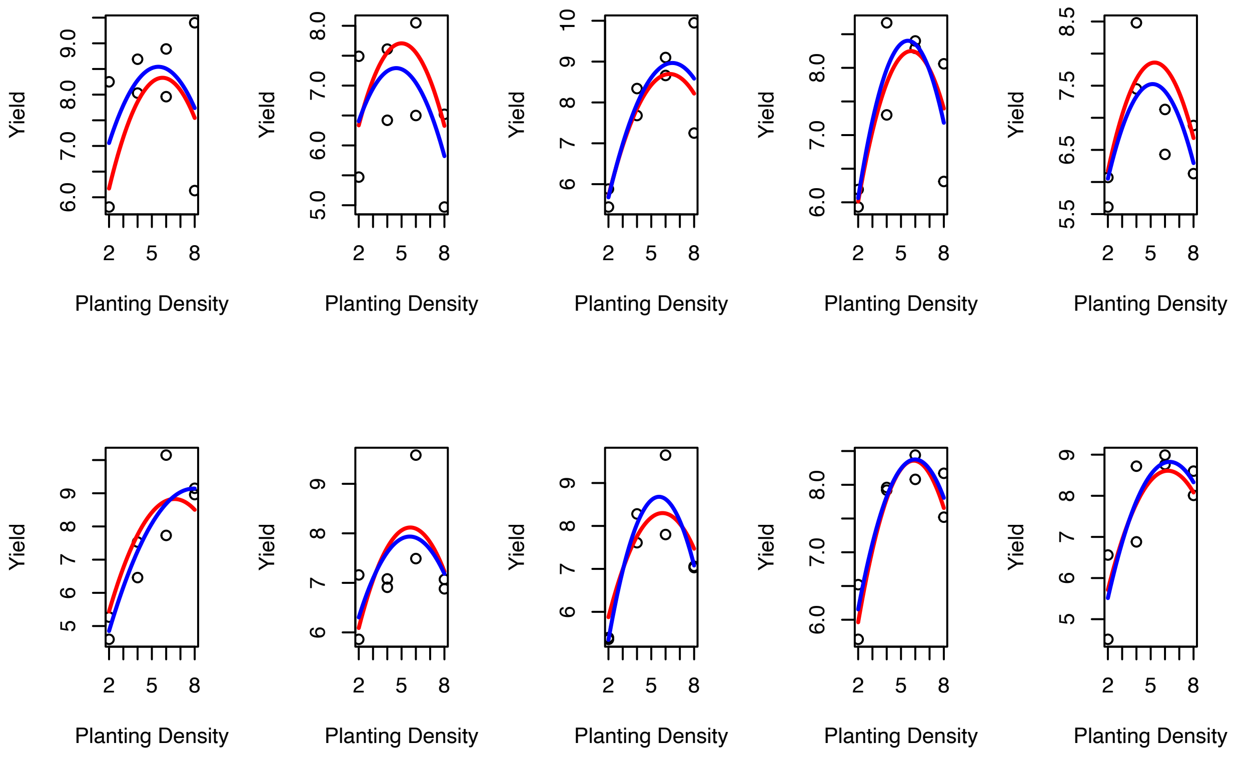
b)

```
ids <- sort(unique(Data$plot))
m <- length(ids);  m;  # Number of plots is m = 10
```

```
## [1] 10
```

```
y <- list();  X <- list();  n <- NULL;

for(j in 1:m)
{
  y[[j]] <- Data[Data$plot == ids[j], 3]
  n[j] <- sum(Data$plot == ids[j])
  x.j <- Data[Data$plot == ids[j], 2]
  x.j2 <- x.j * x.j
  X[[j]] <- cbind(rep(1, n[j]), x.j, x.j2)
}


# Bayesian hierarchical linear regression model

# Set prior parameters here (unit information prior)

p <- 3

mu.0 <- mean_coefs

Lambda.0 <- cov_coefs

S.0 <- cov_coefs;  eta.0 <- p + 2;

sigma2.0 <- sigma_squared_hat;  nu.0 <- 1;

m <- length(models)


# We will run the Gibbs sampler for 10,000 scans, but only save every
#  10th update

S <- 1000;  T <- 10;

sigma2.chain <- rep(NA, S);
```

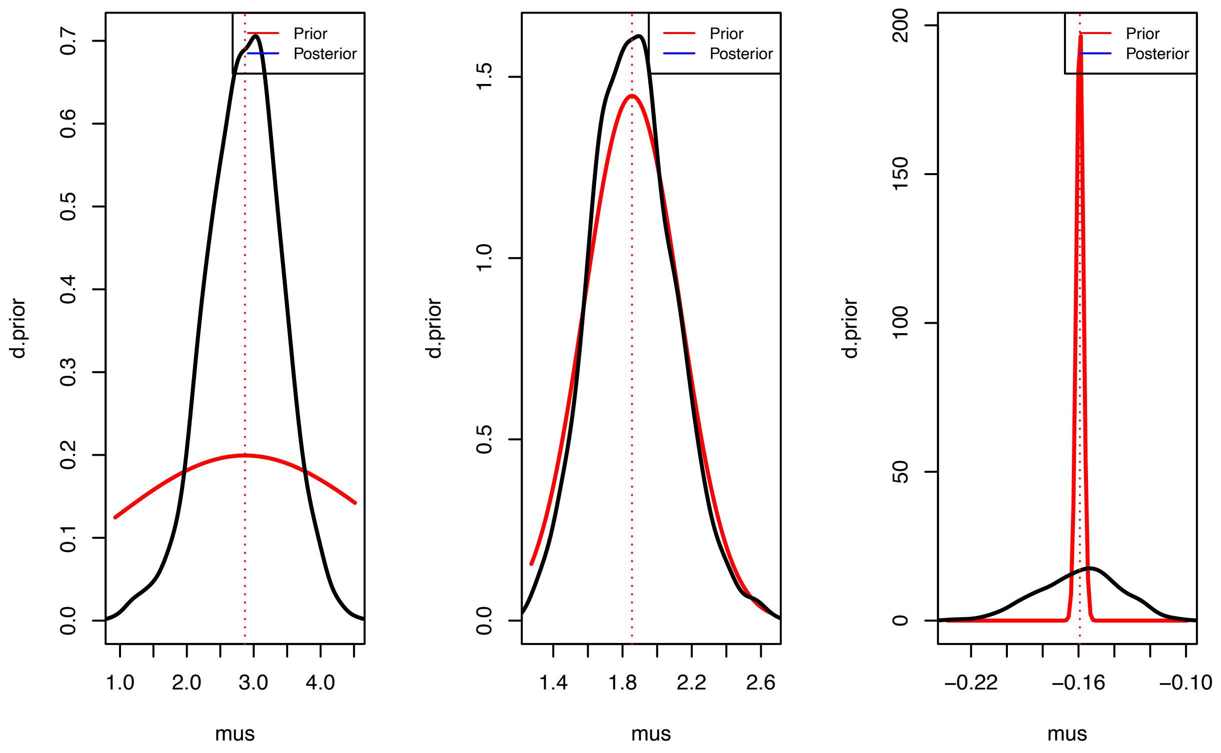
```r
mu.chain <- matrix(NA, S, p)

Sigma.chain <- matrix(NA, S, p^2)

beta.chain <- matrix(NA, S, m*p)


# Compute inverse matrix once and not 10,000 times

Lambda0.inv <- solve(Lambda.0)

Lambda.inv.mu.0 <- Lambda0.inv %*% mu.0


# Starting values for Gibbs sampler

mu <- mu.0;  sigma2 <- sigma2.0;

beta <- coefficients_matrix;   # beta is m x p

Sigma.inv <- solve(S.0);


# Now run it!

run.time <- proc.time()

for(s in 1:S)
{
 for(t in 1:T)
 {
  ###
  # Update the beta_j
  ###
  for(j in 1:m)
  {
   V.j <- solve( Sigma.inv + t(X[[j]]) %*% X[[j]] / sigma2 )
   m.j <- V.j %*% ( Sigma.inv %*% mu + t(X[[j]]) %*% y[[j]] / sigma2 )
   beta[j,] <- rmvnorm(1, mu=as.vector(m.j), sigma=V.j)[1,]
  }
  ###
  # Update mu
  ###
  Lambda.m <- solve( Lambda0.inv + m * Sigma.inv )
  mu.m <- Lambda.m %*% (Lambda.inv.mu.0 + Sigma.inv %*% apply(beta,2,sum))
  mu <- rmvnorm(1, mu=mu.m, sigma=Lambda.m)[1,]
  ###
  # Update Sigma matrix
  ###
  mu.mat <- matrix(mu, m, p, byrow=T)
  S.mu <- t(beta - mu.mat) %*% (beta - mu.mat)
  Sigma.inv <- rWishart(1, eta.0+m, solve(S.0 + S.mu))[,,1]
  ###
  # Update sigma2
  ###
  SSR <- 0
  for(j in 1:m){ SSR <- SSR + sum( (y[[j]] - X[[j]] %*% beta[j,])^2 ) }
  sigma2 <- (nu.0*sigma2.0 + SSR) / rchisq(1, df=nu.0+sum(n))
 } # End of t-loop; at every t-th scan we'll save the results
 sigma2.chain[s] <- sigma2;
 mu.chain[s,] <- mu
 Sigma.chain[s,] <- solve(Sigma.inv)
 for(j in 1:m){ beta.chain[s, seq(j, j+(p-1)*m, m)] <- beta[j,] }
}

# now, plots

# Extract beta samples
beta1 = beta.chain[, 1:10]
beta2 = beta.chain[, 11:20]
beta3 = beta.chain[, 21:30]

# Combine into matrix
beta.mat <- cbind(beta1, beta2, beta3)
beta.post <- colMeans(beta.mat)

# Extract beta samples into matrix with 10 rows (groups) and 3 columns
beta.mat <- matrix(beta.post, ncol = 3, byrow = TRUE)

# Set up x values to plot over
x <- seq(min(Data$density), max(Data$density), length.out = 100)

# Plot curves
par(mfrow = c(2,5))
for(i in 1:10){

  # Predicted response for this group
  yhat <- beta.post[i] + beta.post[i+10]*x + beta.post[i+20]*x^2

  # Predictions from frequentist LM
  newdata <- data.frame(density = x)
  yhat.lm <- predict(models[[i]], newdata)

  # Plot
  plot(Data$density[Data$plot==i], Data$y[Data$plot==i],
       xlab="Planting Density", ylab="Yield")
  lines(x, yhat, col="red", lwd=2)
  lines(x, yhat.lm, col="blue", lwd=2)

}
```



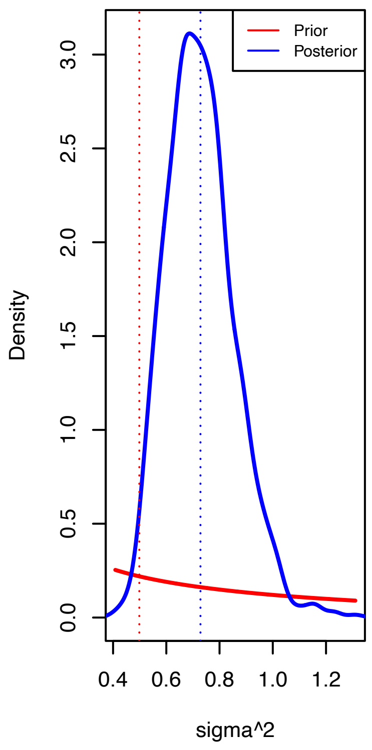
As we can see, the posterior distributions are similar to the prior distributions. However, in some cases, the difference is clear between the two. For example, in plot 2 we can see a big difference between the posterior in red and the prior in blue. The posterior has more information, so it unifies much more the curves in the posterior plots for all crops.

c)

```r
# Prior and posterior for mu
par(mfrow=c(1,3))
for(i in 1:3){

  # Posterior
  den.post <- density(mu.chain[,i])

  # Prior
  mus <- seq(min(mu.chain[,i]), max(mu.chain[,i]), length.out=100)
  d.prior <- dnorm(mus, mu.0[i], Lambda.0[i,i])

  # Plot
  plot(mus, d.prior, type="l", lwd=2, ylim=c(0, max(d.prior, den.post$y)),
       col="red")
  lines(den.post, lwd=2)

  abline(v=mu.0[i], col="red", lty="dotted")
  legend("topright", legend=c("Prior", "Posterior"),
       col=c("red", "blue"), lty=1, cex=0.8)
}
```



```r
# Prior and posterior for sigma2
# Sample values for x-axis
x = seq(min(sigma2.chain), max(sigma2.chain), length.out=1000)
# Prior density
y = sigma2.0 / (nu.0 / dchisq(x, df= nu.0))
# Posterior density
den <- density(sigma2.chain)

# Plot
plot(x, y, type="l", ylim = c(0, max(den$y)), lwd=2, col="red",
     ylab="Density", xlab="sigma^2")
lines(den, col="blue", lwd=2)
# Add vertical lines
abline(v = sigma2.0, lty="dotted", col="red")
abline(v = mean(sigma2.chain), lty="dotted", col="blue")
legend("topright", legend=c("Prior", "Posterior"),
       col=c("red", "blue"), lty=1, cex=0.8)
```



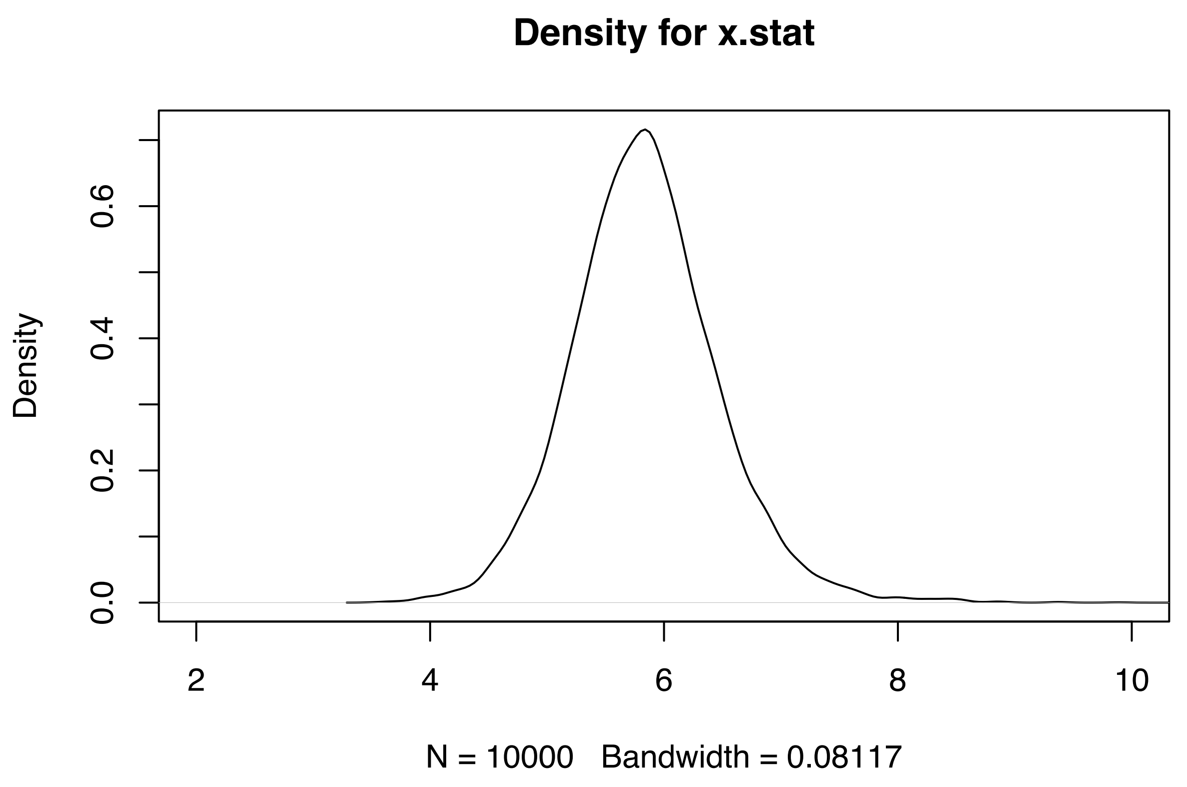
d)

i)

```
#given that all betas are next to each other in my beta chain
#i will rbind all beta2 and beta3 values
beta.1.post <- rbind(beta.chain[,1:10])
beta.2.post <- rbind(beta.chain[,11:20])
beta.3.post <- rbind(beta.chain[,21:30])

#length should be 10,000 since we are taking into account all plots
length(beta.2.post)
```

```
## [1] 10000
```

```
# calculate the max
x.stat <- -beta.2.post / (2 * beta.3.post)

plot(density(x.stat), main = "Density for x.stat", xlim = c(2,10))
```



```
#ci
quantile(x.stat, probs = c(.025, .975))
```

```
##     2.5%    97.5%
## 4.673899 7.158894
```

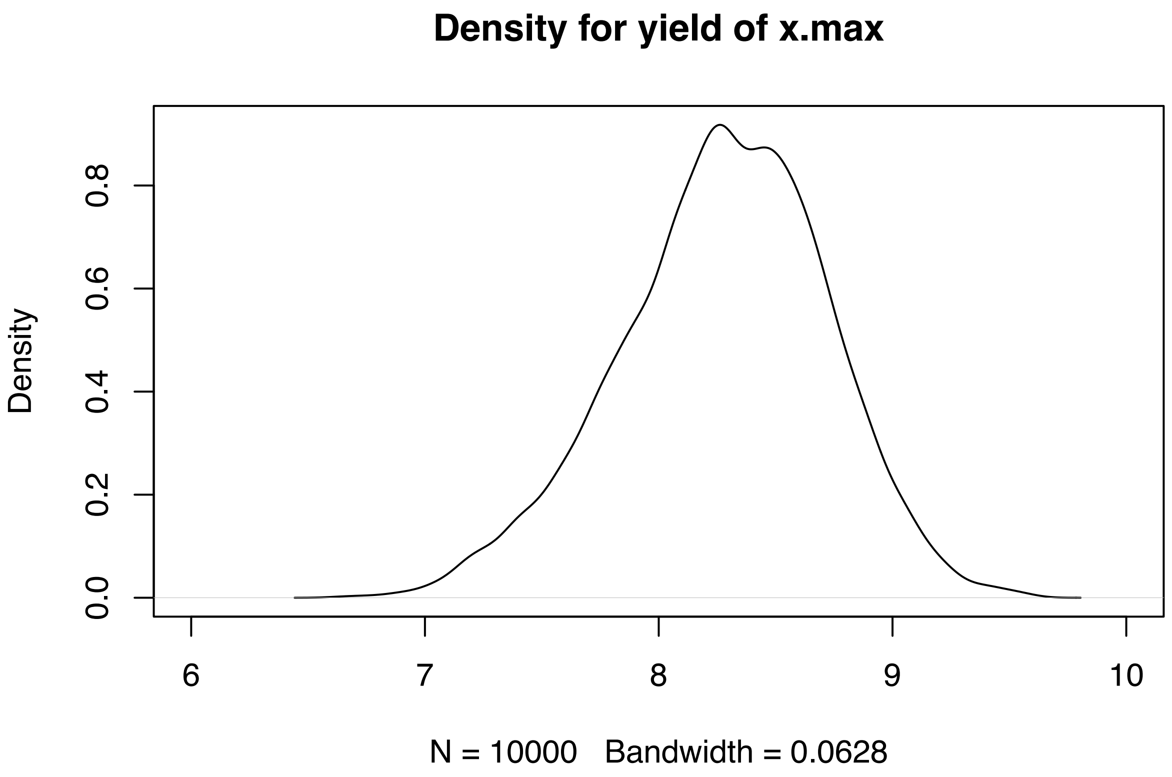
ii)

```
# Compute density
density.xstat <- density(x.stat)

# Mode is value at maximum density
mode <- density.xstat$x[which.max(density.xstat$y)]
paste0("The maximum value is located at x = ", mode)
```

```
## [1] "The maximum value is located at x = 5.83914656635812"
```

iii)

```
yield.ppd <- beta.1.post + beta.2.post * mode + beta.3.post * mode^2
plot(density(yield.ppd), main = "Density for yield of x.max", xlim = c(6,10))
```



From the analysis of the optimal plotting density we learned that it was around 6. This is based on our prior belief and our observations, even though in this case our prior belief was informed by our observations!

From the prediction interval for the yield, we learned that it would most likely be between 7 to 9. This again follows from our analysis of the beta coefficients. This result is derived from an analysis of the pooled coefficients for all plots.

Question 3

a)

```r
rm(list=ls())
library(rstan)
```

```
## Loading required package: StanHeaders

##
## rstan version 2.32.3 (Stan version 2.26.1)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```r
library(mcmcse)
library(coda)
```

```
##
## Attaching package: 'coda'

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

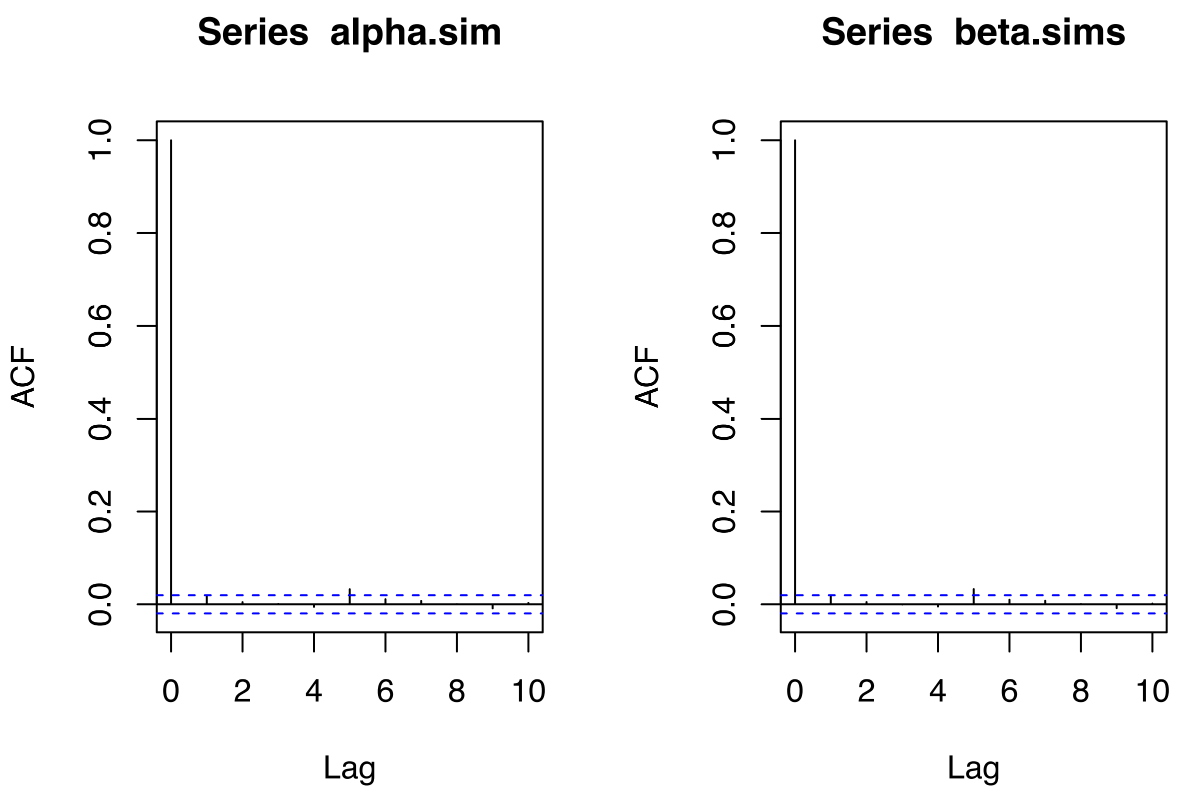
```r
file <- "http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/msparrownest.dat"
Data <- read.table(file=file, header=T)
rm(file)
colnames(Data) <- c("y", "x")
y <- Data$y
x <- Data$x

stan_model <- "
 data{
  int<lower=0> n;
  int<lower=0, upper=1> y[n];
  vector[n] x; }
 parameters{
  real alpha;
  real beta;   }
 model{
  alpha ~ normal(0,10);
  beta  ~ normal(0, 2);
  y ~ bernoulli_logit(alpha + beta*x); } "
```

```r
fit_stan <- stan(model_code=stan_model,
  data=list(n=length(y), y=y, x=x),
  chains=1, iter=11000, warmup=1000)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: ‘Apple clang version 15.0.0 (clang-1500.0.40.1)’
## using SDK: ‘’
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/Core:96
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.4e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.24 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:     1 / 11000 [  0%]  (Warmup)
## Chain 1: Iteration:  1001 / 11000 [  9%]  (Sampling)
## Chain 1: Iteration:  2100 / 11000 [ 19%]  (Sampling)
## Chain 1: Iteration:  3200 / 11000 [ 29%]  (Sampling)
## Chain 1: Iteration:  4300 / 11000 [ 39%]  (Sampling)
## Chain 1: Iteration:  5400 / 11000 [ 49%]  (Sampling)
## Chain 1: Iteration:  6500 / 11000 [ 59%]  (Sampling)
## Chain 1: Iteration:  7600 / 11000 [ 69%]  (Sampling)
## Chain 1: Iteration:  8700 / 11000 [ 79%]  (Sampling)
## Chain 1: Iteration:  9800 / 11000 [ 89%]  (Sampling)
## Chain 1: Iteration: 10900 / 11000 [ 99%]  (Sampling)
## Chain 1: Iteration: 11000 / 11000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.107 seconds (Warm-up)
## Chain 1:                0.717 seconds (Sampling)
## Chain 1:                0.824 seconds (Total)
```

```
## Chain 1:
```

```
alpha.sim <- extract(fit_stan)$alpha
beta.sims <- extract(fit_stan)$beta

#lets look at the acf plots
par(mfrow=c(1,2))
acf(alpha.sim, lag.max = 10)
acf(beta.sims, lag.max = 10)
```



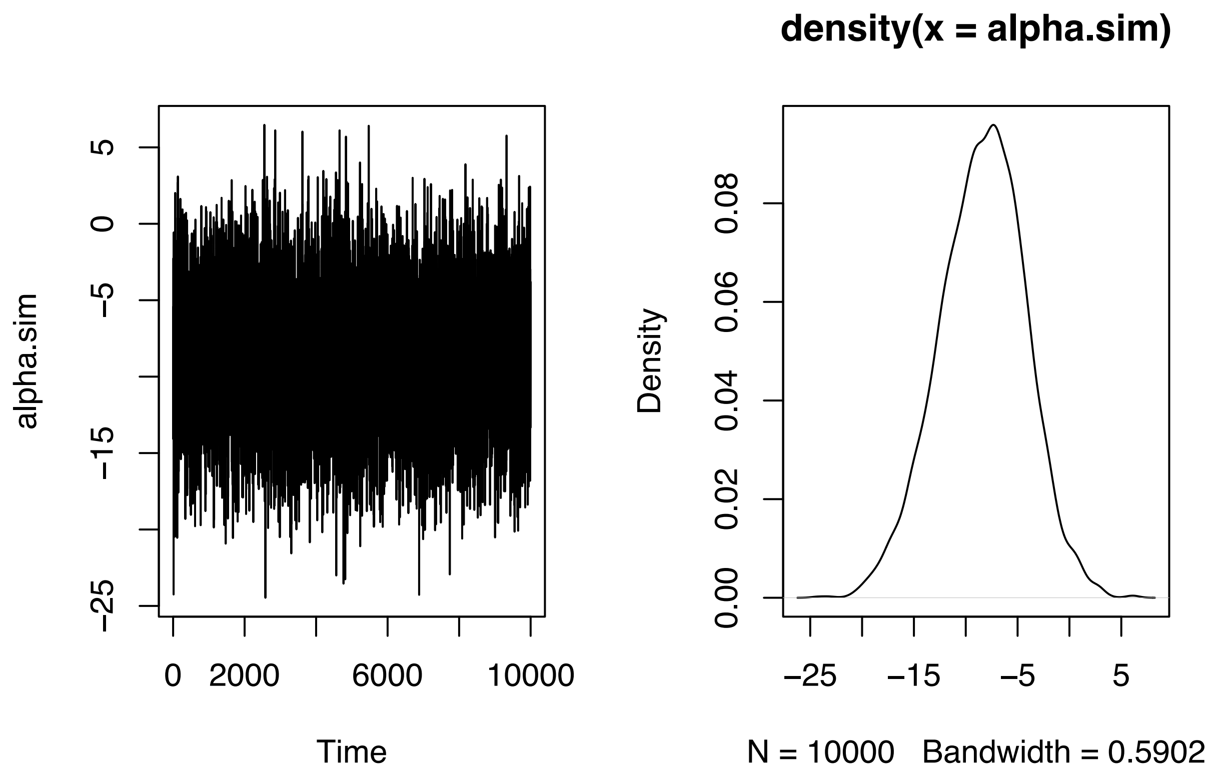
```
# it looks almost uncorrelated!! effective sample sizes should be
# 10,000 or close
ess(as.matrix(alpha.sim))
```

```
## [1] 10000
```

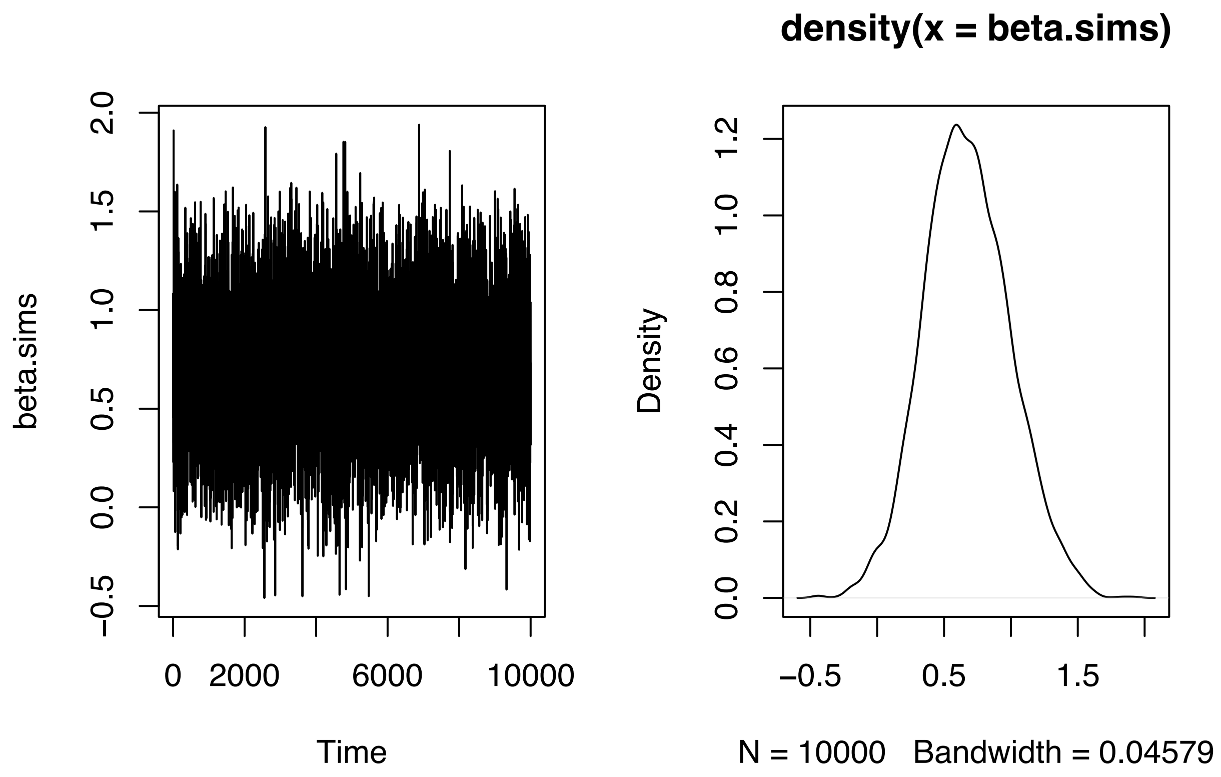
```
ess(as.matrix(beta.sims))
```

```
## [1] 10000
```

```
# now plot trace plot and density for alpha
par(mfrow=c(1,2))
ts.plot(alpha.sim)
plot(density(alpha.sim))
```

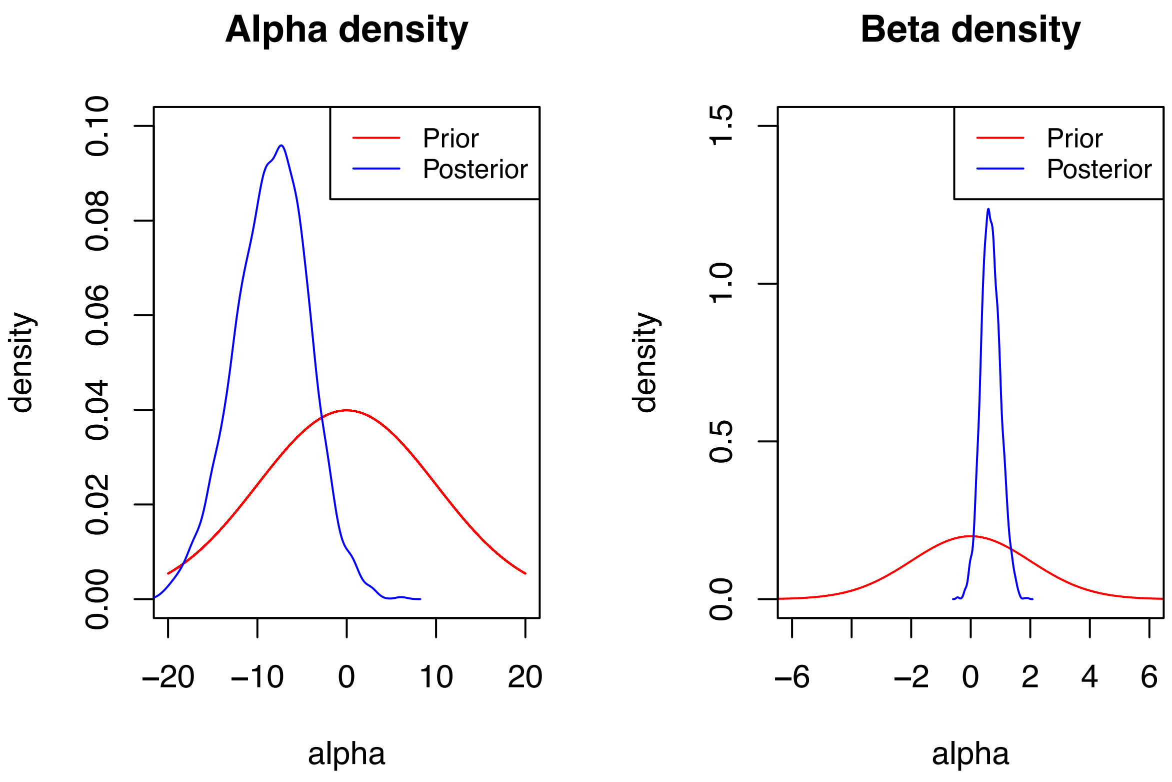


```
# now plot trace plot and density for beta
par(mfrow=c(1,2))
ts.plot(beta.sims)
plot(density(beta.sims))
```



b)

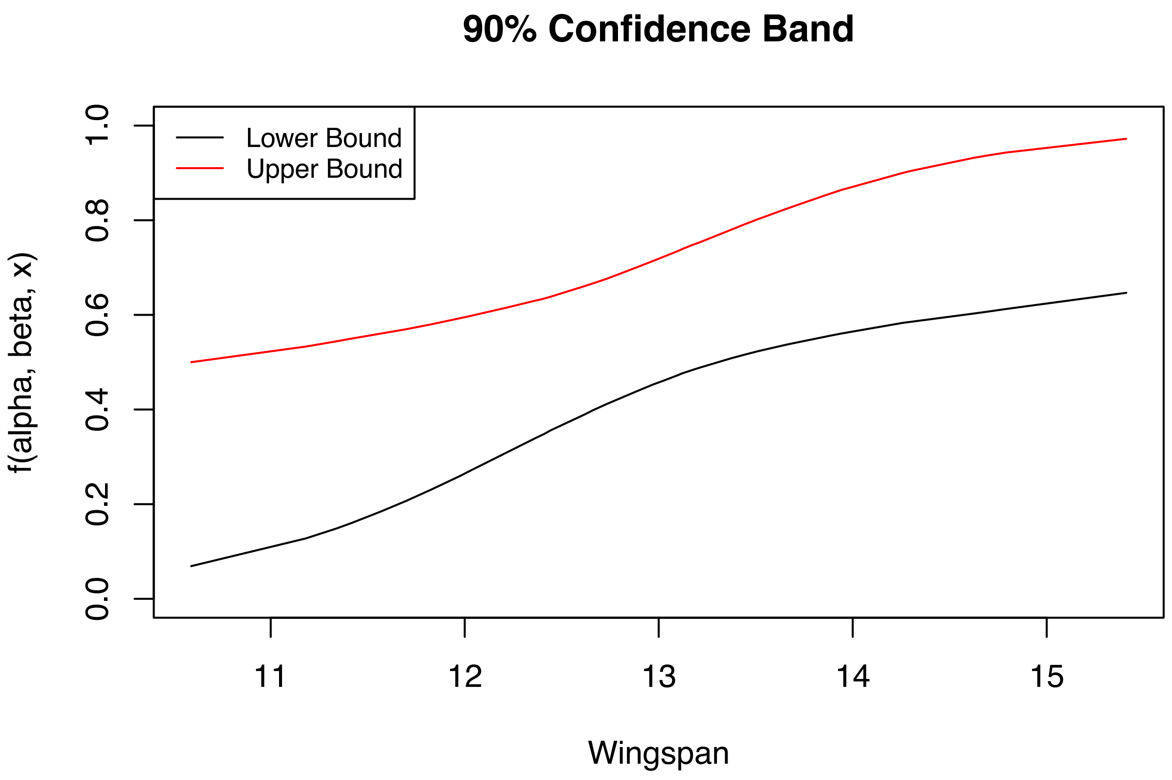
```r
x <- seq(-20, 20, .001)
y <- dnorm(x, 0, sd = 10)
y2 <- dnorm(x, 0, sd = 2)

# now plot prior and posterior of both
par(mfrow=c(1,2))
plot(x,y, type="l", col = "red", ylim = c(0, .10),
     xlab = "alpha", ylab = "density", main = "Alpha density")
lines(density(alpha.sim), col="blue")
legend("topright", legend=c("Prior", "Posterior"),
       col=c("red", "blue"), lty=1, cex=0.8)

plot(x,y2, type="l", col = "red", ylim = c(0, 1.5), xlim = c(-6, 6),
     xlab = "alpha", ylab = "density", main = "Beta density")
lines(density(beta.sims), col="blue")
legend("topright", legend=c("Prior", "Posterior"),
       col=c("red", "blue"), lty=1, cex=0.8)
```



c)

```r
# Define function
f = function(alpha, beta, x) {
  exp(alpha + beta*x) / (1 + exp(alpha + beta*x))
}
```

```r
# Get data
x <- Data$x

# Initialize matrix of predictions
n_x <- length(x)
n_sim <- length(alpha.sim)
f_preds <- matrix(NA, n_x, n_sim)

# Populate prediction matrix
for(i in 1:n_sim){
  for(j in 1:n_x){
    f_preds[j, i] <- f(alpha.sim[i], beta.sims[i], x[j])
  }
}

# Compute confidence interval bounds
lower <- apply(f_preds, 1, quantile, 0.05)
upper <- apply(f_preds, 1, quantile, 0.95)

# Plot
plot(sort(x), sort(lower), type="l", ylim = c(0,1),
     xlab = "Wingspan", ylab = "f(alpha, beta, x)",
     main = "90% Confidence Band")
lines(sort(x), sort(upper), col="red", type = "l")
legend("topleft", legend=c("Lower Bound", "Upper Bound"),
       col=c("black", "red"), lty=1, cex=0.8)
```



In this model, the confidence band represents the probability of a male sparrow nesting given its wingspan. This allow us to do inference using our MCMC output and simulating possible probabilities of male sparrows