Lab 06 - MCMC

GE 509

October 14, 2014

The goal of this week's lab is to get "under the hood" of Bayesian numerical methods in order to understand what is going on behind the scenes in BUGS, where output just magically appears. It is also important to be able to fall back on doing these computations explicitly because we will occasionally find ourselves faced with models that are too complex or too large for BUGS. For this week's lab we will return to using R.

Bayesian Regression using Gibbs Sampling

Linear regression is a good place to start our exploration of numerical methods because it is the foundation for the large majority of the data analysis that occurs using classical methods (recall that ANOVA models are just a special case of regression) and because it gives us a foundation to build off of for exploring more complex models.

Recall from lecture 13 and chapter 7.4 of the textbook that the standard Bayesian regression model assumes a Normal likelihood, a Normal prior on the regression parameters, and an Inverse Gamma prior on the variance.

$$P(b, \sigma^2|X, y) \propto N_n(y|Xb, \sigma^2 I)N_p(b|b_0, V_b)IG(\sigma^2|s_1, s_2)$$

Within the Gibbs sampler we will be iteratively sampling from each of the conditional posterior distributions:

The regression parameters given the variance

$$P(b|\sigma^2, X, y) \propto N_n(y|Xb, \sigma^2 I)N_p(b|b_0, V_b)$$

The variance given the regression parameters

$$P(\sigma^2|b, X, y) \propto N_p(b|b_0, V_b)IG(\sigma^2|s_1, s_2)$$

We can divide the R code required to perform this analysis into three parts:

- Code used to set-up the analysis
  - load data
  - specify parameters for the priors
  - set up variables to store MCMC
  - specify initial conditions
- MCMC loop
- Code used to evaluate the analysis
  - Convergence diagnostics
  - Summary statistics
  - Credible & predictive intervals

As a reminder, we **strongly** recommend that you assemble the bits of code from the lab into a single Rmd script file. This will make re-running parts of the code much easier since you will not have to cut-and-paste and it provides you with a script to start from if you want to use the code in the future for your own data

analysis. Also recall from Lab 1 that the functions save(), save.image(), and load() allow you save the output of your analysis and reload it later.

Set up

For the first part of this lab we are going to be simulating data from a known model instead of using a real data set. This exercise serves two purposes – first, to allow us to evaluate the MCMC's ability to recover the "true" parameters of the model and second to demonstrate how "pseudodata" can be generated. Pseudo-data "experiments" can be useful for assessing the power of a test/statistical model and for exploring experimental design issues (e.g. how many samples do I need, how should I distribute them?). In a real analysis you would substitute code that loads up your data from a file instead of the following bit of code

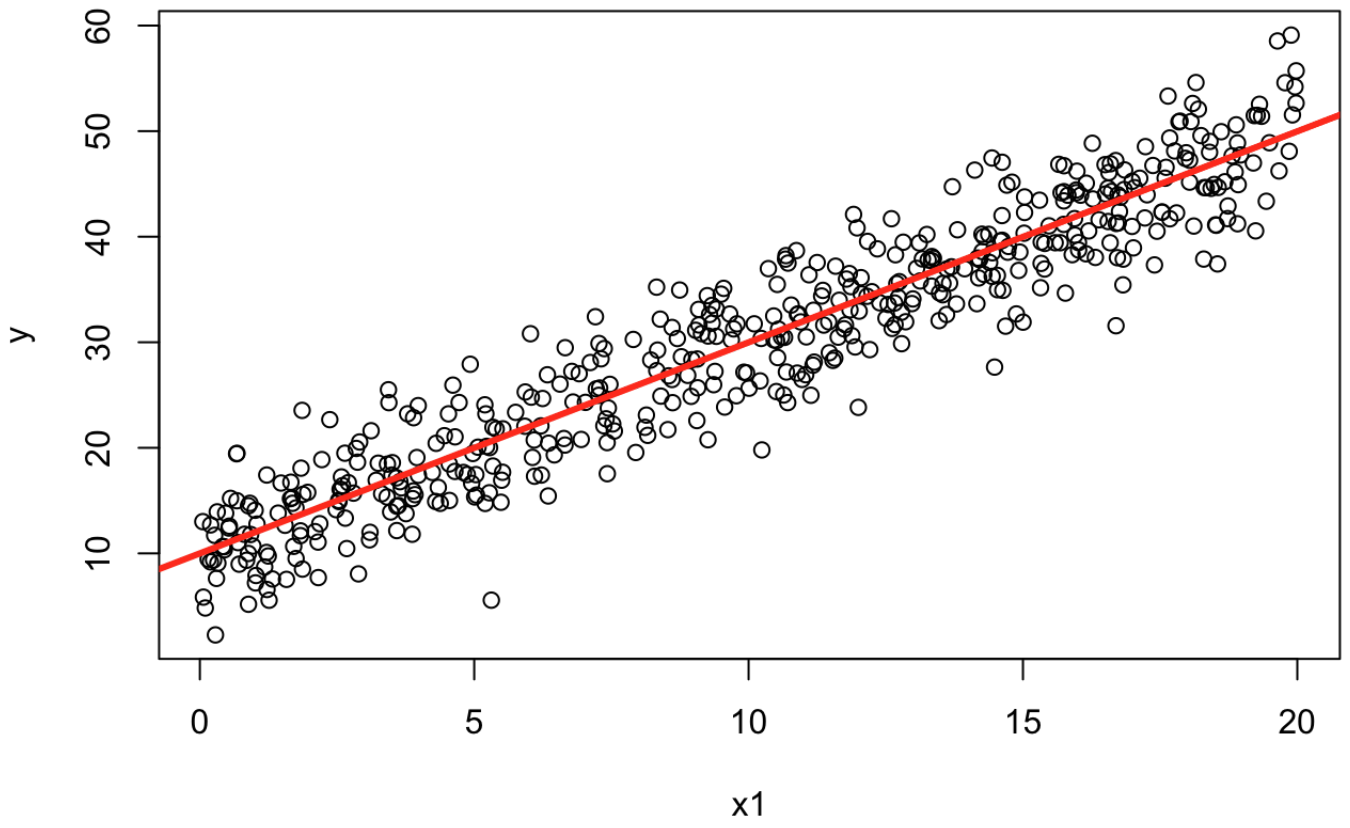
```
### Part 1: simulate data from a known model
n <- 500            ## define the sample size
b0 <- 10                ## define the intercept
b1 <- 2                 ## define the slope
beta <- matrix(c(b0,b1),2,1)        ## put "true" regression parameters in a matrix
sigma2 <- 4^2               ## define the variance (s.d. = 4)
```

In this first part we define the true parameters of our model that we're going to simulate from. In the next block we will first simulate our covariate, x1, by assuming that it is uniformly distributed over the range that we're interested in (0,20) and then create our design matrix, x, which is a matrix with the first column of all 1's and the second column being our covariate pseudodata. To combine these two columns we use the *cbind* command, which creates a matrix assuming that the vectors passed to the function are the columns of the matrix. When the design matrix, x, is multiplied by the parameter vector, beta, the first column is multiplied by the first element in the beta vector, b1, which is the intercept. The second column in the design matrix, which is our covariate x, is multiplied by the second beta, b1, the slope. Together this gives the calculation we desire, b0 + b1*x1, which is our linear regression model. This calculation is also the expected value of the response variable, y, at that x1.

```
x1 <- runif(n,0,20)
x <- cbind(rep(1,n),x1)
y <- matrix(rnorm(n,x%*%beta,sqrt(sigma2)),n,1)
```

As you can see, in the third line of this code we generate pseudodata for the response variable, y, by taking n samples from a random normal distribution that has a mean that is the expected value of our regression model and the standard deviation we specified above. This pseudodata is then arranged in a matrix that has n rows and 1 column. Once we've generated our pseudodata lets do some "exploratory data analysis" to look at the data we've generated and the model that it came from

```
plot(x1,y)
abline(b0,b1,col=2,lwd=3)
```



Now that we have "data", the next task in setting up the analysis is to specify the parameters for the prior distributions. Here we'll assume that beta has a Normal prior with mean bprior and a variance Vb. Since beta is a vector, the prior mean, bprior, is also a vector, and the prior variance is a matrix. Since we have no prior conceptions about our data we'll select relatively weak priors, assuming a prior mean of 0 and a prior variance matrix that is diagonal (i.e. no covariances) and with a moderately large variance of 1000 (s.d. of 31.6). We'll use the *diag* command to set up a diagonal matrix that has a size of 2 and values of 1000 along the diagonal. In practice we never actually need to use the prior variance matrix, but we will frequently use the inverse of the matrix. Therefore we will compute the inverse of the prior variance matrix, vinvert, using the *solve* function. Finally we will specify an uninformative inverse gamma prior on the variance with parameters s1 = 0.1 and s2 = 0.1.

```
## specify priors
bprior <- as.vector(c(0,0))
vinvert <- solve(diag(1000,2))
s1 <- 0.1
s2 <- 0.1
```

Next we'll pre-compute some quantities that are frequently used in the Gibbs sampler. By computing these quantities only once rather than recomputing every step of the MCMC we can speed up the algorithm, especially for matrix multiplication on large data sets – matrix multiplication is an $O(n^3)$ algorithm, meaning that runtime increases as the cube of the size of the matrix.

```
##precompute frequently used quantities
XX <- t(x) %*% x
XY <- t(x) %*% y
VbB <- vinvert %*% bprior
```

Next, we'll want to load up a number of R libraries that we'll use in the analysis. The first, coda, provides a lot of the same MCMC diagnostic functions that we used in BUGS. The second, mvtnorm, provides an implementation of the multivariate normal, which is not a default function in R.

```
##load libraries
library(coda)
```

```
## Loading required package: lattice
```

```
library(mvtnorm)
```

If trying to load these libraries gives an error the most likely problem is that they are not installed on your system. To download them click on Packages > Install. You only have to download the package once, but you do need to load it into R using the "library" command every time you start R and want to use that package.

Next, we'll define some variables that control the MCMC itself and provide storage for the values we generate.

```
##storage for MCMC
ngibbs <- 10                ## number of updates
bgibbs <- matrix(0.0,nrow=ngibbs,ncol=2)    ## storage for beta
sgibbs <- numeric(ngibbs)           ## storage for sigma2
```

You'll note above that the number of MCMC updates seems very small. It is generally a good idea to start with a small sample to make sure the code works, and then reset this variable to a larger number once you're sure things are behaving well. Finally, the last thing we need to do to prep for the MCMC loop is to specify the **initial conditions**. In this case we'll only specify an initial condition for the variance. The first step of the Gibbs sampler will then draw the first value for beta conditioned on this value for the variance. We'll also define the inverse of the variance since this quantity is frequently used in computation.

```
## initial conditions
sg <- 50
sinv <- 1/sg
```

MCMC loop

The core of this analysis is the main loop of the MCMC where we will iteratively draw from the posterior distribution for the regression parameters conditioned on the variance and the variance conditioned on the regression parameters. All of this occurs within a large loop that counts the number of iterations and stores the current value of the parameter values. We'll begin with the R code for the overall structure of the MCMC and then fill in the details for each sampler

```
## Gibbs loop
for(g in 1:ngibbs){

  ## sample regression parameters
<<insert normal here >>

  ## sample variance
<< insert inverse gamma here >>

  ## storage
  bgibbs[g,] <- b  ## store the current value of beta vector
  sgibbs[g]  <- sg  ## store the current value of the variance

  if(g %%100 == 0) print(g) ##show how many steps have been performed
}
```

Recall from lecture 13 and from Section 7.4 in the textbook that conditional posterior for the regression parameters

$$P(b|\sigma^2, X, y) \propto N_n(y|Xb, \sigma^2 I)N_p(b|b_0, V_b)$$

has a multivariate normal posterior that takes on the form

$$p(b|\sigma^2, X, y) \propto N_p(b|Vv, V)$$

where

$$V^{-1} = \sigma^{-2}X^T X + V_b^{-1}$$

$$v = \sigma^{-2}X^t y + V_b^{-1}b_0$$

We can implement this sampler in R as

```
## sample regression parameters
 bigV    <- solve(sinv*XX + vinvert)  ## Covariance matrix
 littlev <- sinv*XY + VbB
 b = t(rmvnorm(1,bigV %*% littlev,bigV))   ## Vv is the mean vector
```

where **rmvnorm** is the multivariate version of rnorm that takes a mean vector and a covariance matrix as it's 2nd and 3rd arguments. We'd recommend taking a look at the output of each step of these calculations to get a feel for what is being calculated. This code should be cut-and-pasted into the overall MCMC loop above.

Next lets look at the sampler for the variance term, which has a posterior distribution

$$P(\sigma^2|b, X, y) \propto N_p(b|b_0, V_b)IG(\sigma^2|s_1, s_2)$$

that takes on an Inverse Gamma posterior

$$IG(\sigma^2|u_1, u_2) \propto \left(\sigma^2\right)^{-(u_1+1)} exp\left[\frac{-u_2}{\sigma^2}\right]$$

where $u_1 = s_1 + n/2$ and $u_2 = s_2 + \frac{1}{2}(y - Xb)^T(y - Xb)$

We can implement this in R as

```
## sample variance
u1 <- s1 + n/2
u2 <- s2 + 0.5*crossprod(y-x%*%b)
sinv <- rgamma(1,u1,u2)
sg <- 1/sinv
```

Since R does not have a built in inverse gamma distribution we instead sample the inverse of the variance (aka the precision) as a gamma, and then compute the variance in the final step. Also note the introduction of the crossprod(A) function that calculates $A^T A$ and is slightly more efficient than doing the calculation explictly as `t(A) %*% A`. This bit of code should also be cut-and-pasted into the overall MCMC loop.

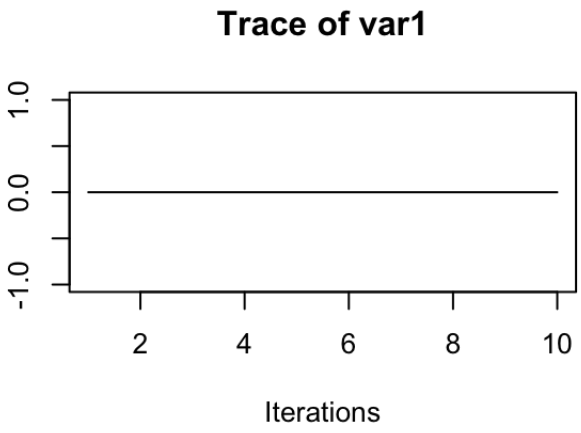
At this point you should run the MCMC loop to check for errors and take a look at the output (bgibbs and sgibbs) to see if values of the parameters make sense. Once you are confident in the results, you should increase ngibbs to a larger values (e.g. ngibbs <- 10000) and re-run the script from that point through the MCMC loop.

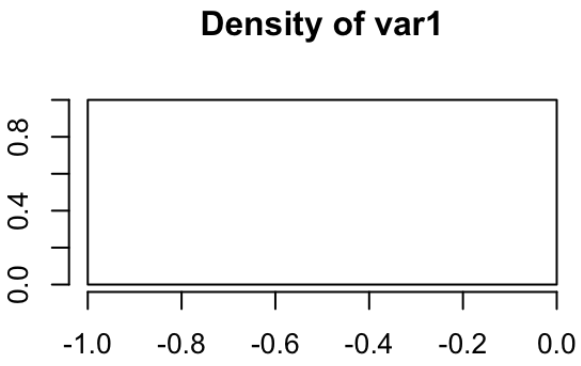
RUNNING MCMC IN R CAN BE SLOW, now might be a good time for a coffee break!

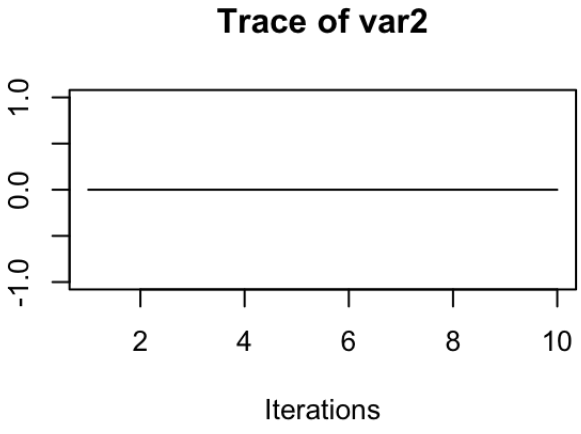
Evaluation

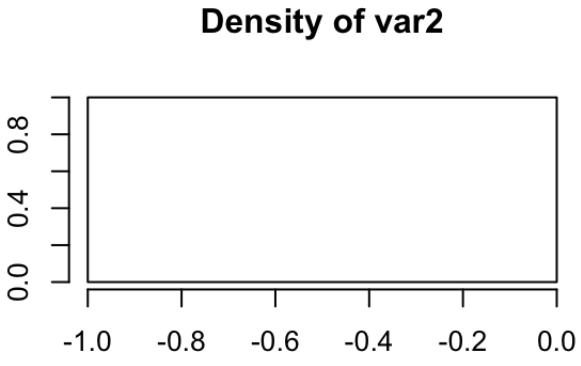
As we did in BUGS, we need to evaluate the output of the MCMC. We will make use of the "coda" library to simplify these graphs and calculations. Let's first look at the output for the regression parameters.

```
## diagnostics of the MCMC
bmcmc <- mcmc(bgibbs)  ## convert to MCMC object
plot(bmcmc)          ## mcmc history and density plot
```

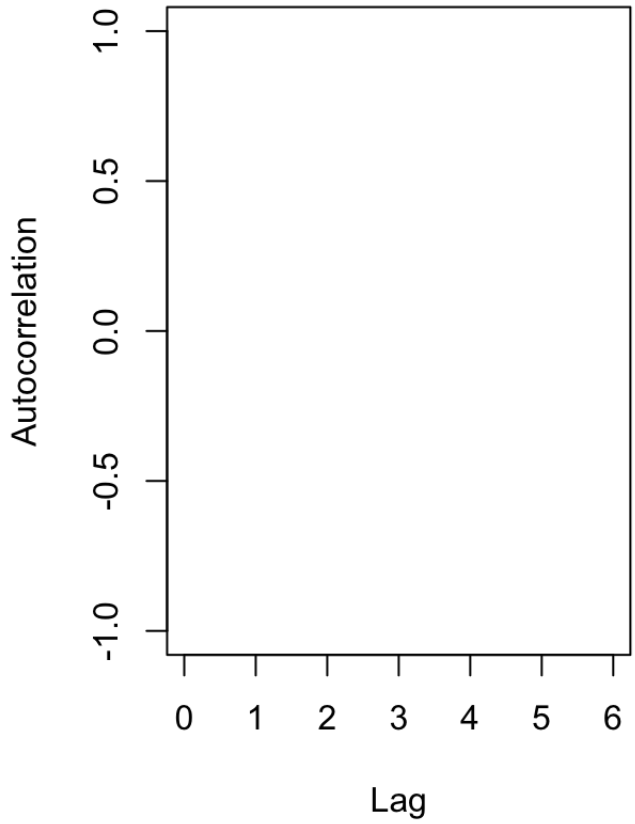


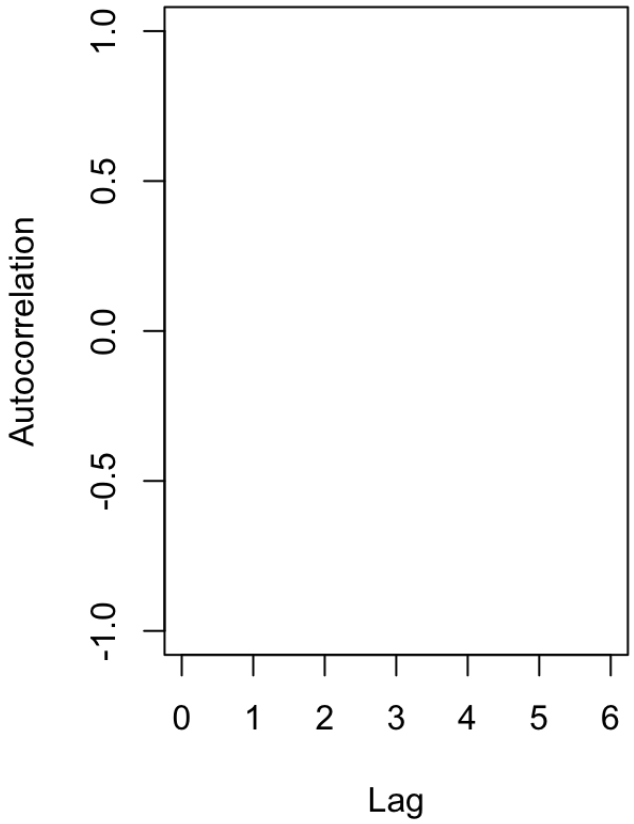




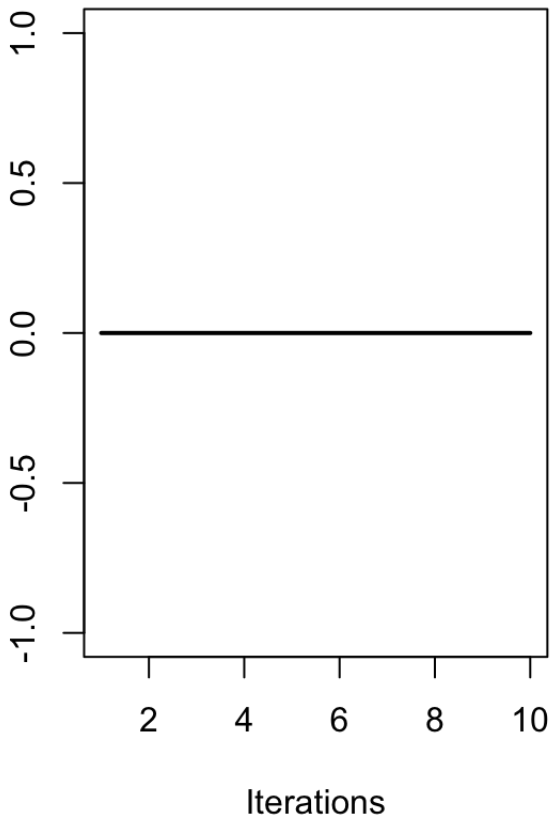


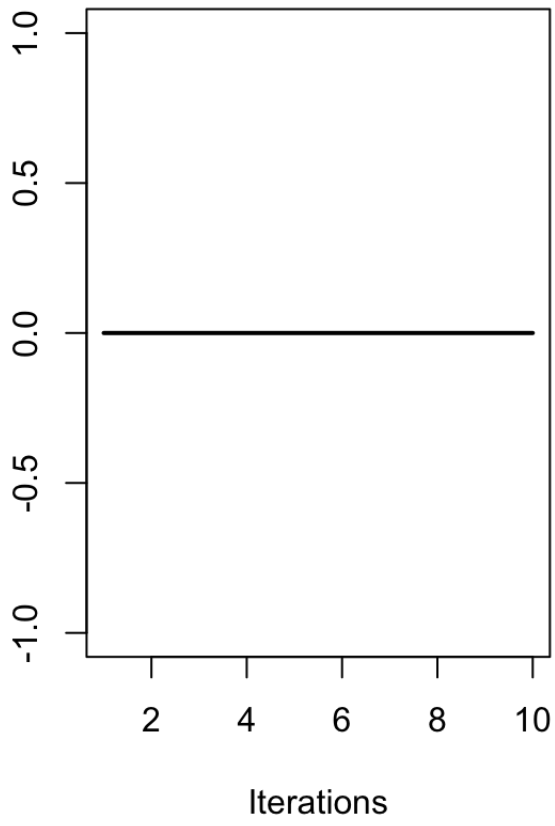

```
autocorr.plot(bmcmc)        ## autocorrelation
```






```
cumuplot(bmcmc)     ## quantile plot
```





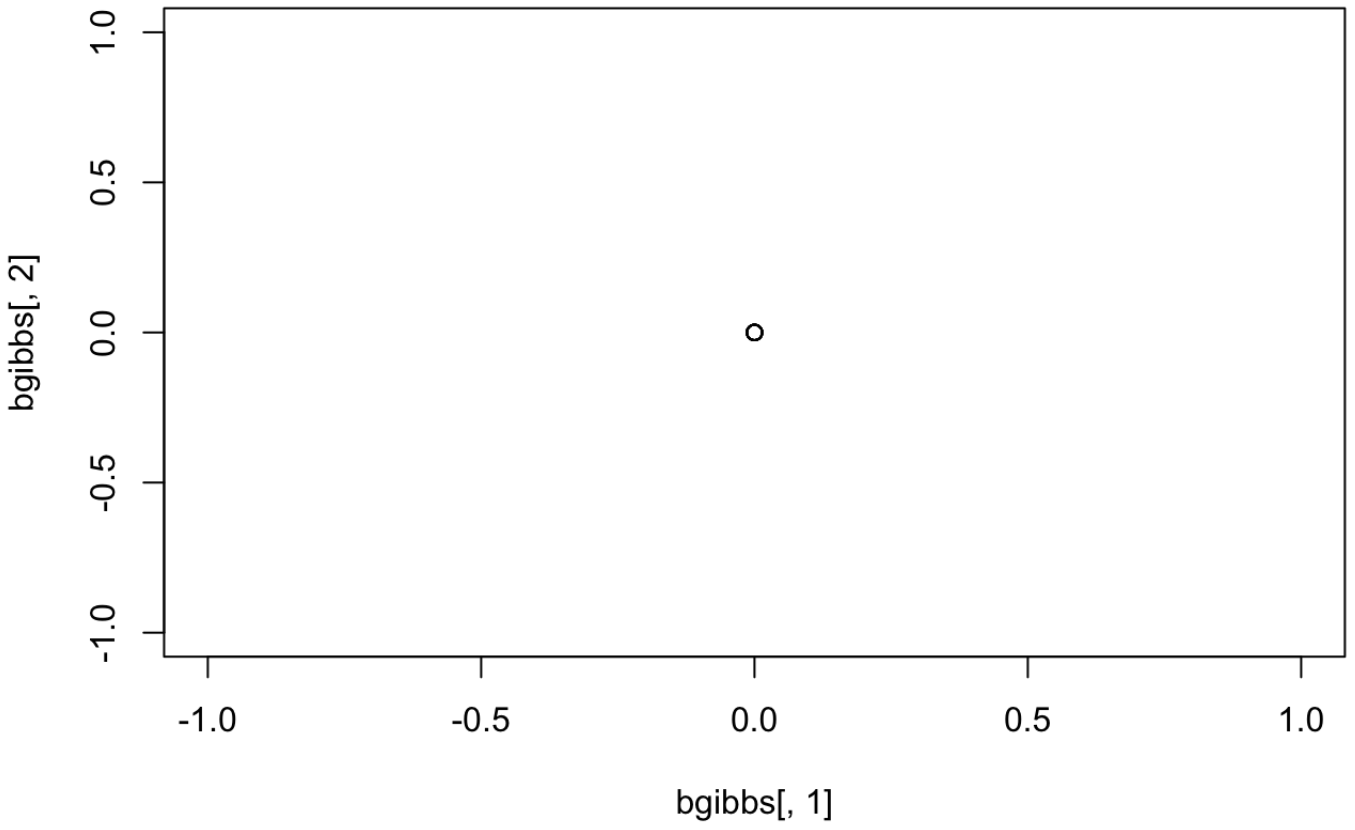

```
1-rejectionRate(bmcmc)  ## acceptance rate
```

```
## var1 var2
##    0    0
```

```
summary(bmcmc)      ## summary table
```

```
##
## Iterations = 1:10
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##      Mean SD Naive SE Time-series SE
## [1,]    0  0        0              0
## [2,]    0  0        0              0
##
## 2. Quantiles for each variable:
##
##      2.5% 25% 50% 75% 97.5%
## var1    0   0   0   0     0
## var2    0   0   0   0     0
```

```
plot(bgibbs[,1],bgibbs[,2]) ## pairs plot to evaluate parameter correlation
```



From these graphs we see that the posterior samples for beta have negligible autocorrelation and burn-in is rapid. If, on the other hand, had we found an initial transient period or significance autocorrelation we can thin the MCMC in R as

```
beg = 1
thin = 1
bmcmc <- mcmc(bgibbs[seq(from=beg,to=ngibbs,by=thin),])
```

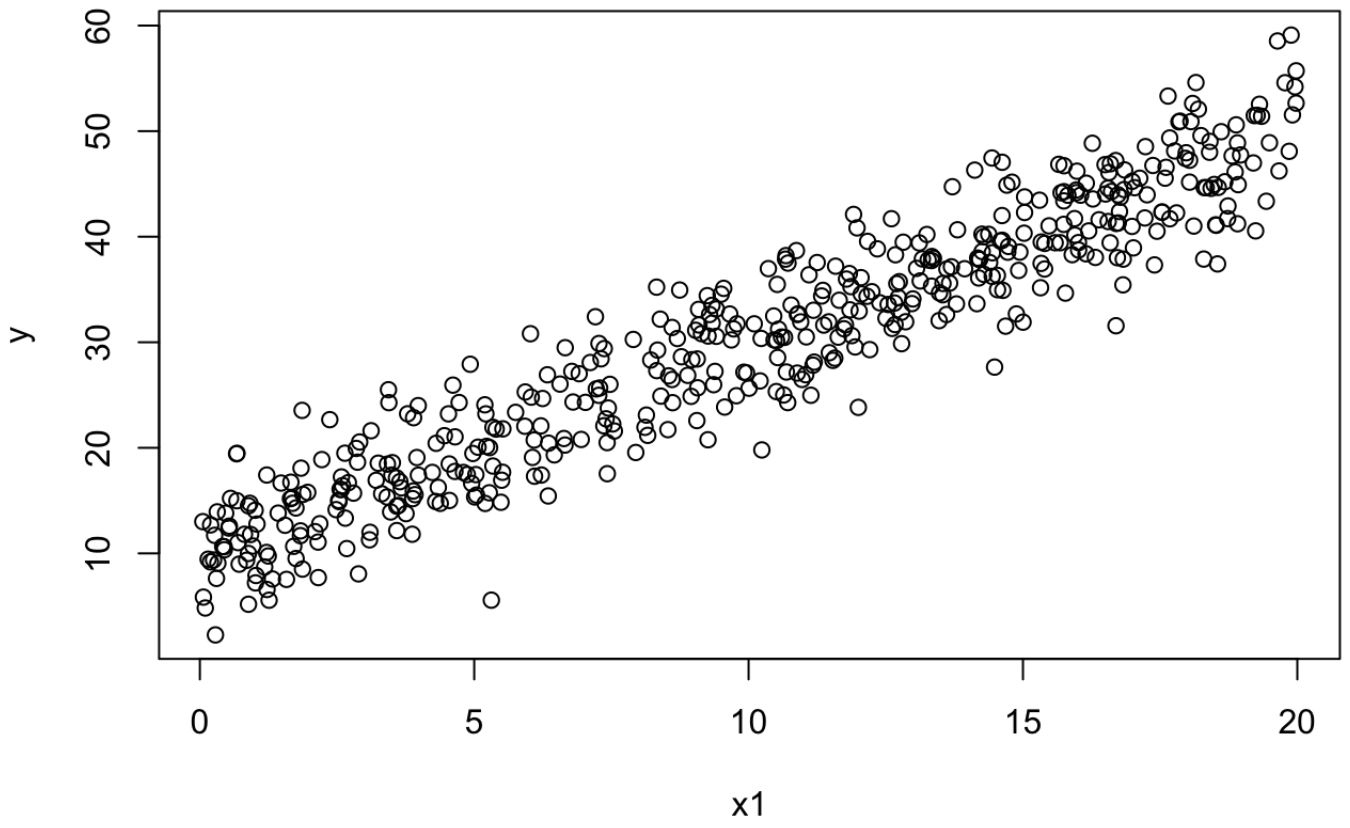
Lab Report Task 1

- Evaluate the MCMC chain for the variance. Include relevant diagnostic plots.
- Compare the summary statistics for the Bayesian regression model to those from the classical regression: summary(lm(y ~ x1)). This should include a comparison of the means and uncertainties of **all 3 model parameters**
- Compare the fit parameters to the "true" parameters we used to generate the pseudo-data. How well does the statistical analysis recover the true model?

Regression Credible Intervals

When fitting the mean of a distribution, the density plot of that distribution and its moments/quantiles provides all the information we need to evaluate the performance of the model. By contrast, when fitting a process model that has covariates, whether it be a simple regression or a complex nonlinear model, we are also often interested in the error estimate on the overall model. This error estimate comes in two forms, the credible interval and the prediction interval. The credible interval, which is the Bayesian analog to the frequentist confidence interval, provides an uncertainty estimate based on the uncertainty in the model parameters. We have already seen Bayesian credible intervals frequently as the quantiles of the posterior distribution of individual parameters. We can extend this to an estimate of model uncertainty by looking at the uncertainty in the distribution of the model itself by calculating a credible interval around the regression line. Numerically we do this by evaluating the model over a sequence of covariate values for every pair of parameter values in the MCMC sequence. Lets begin by looking at a subset of the MCMC

```
xpred <- 0:20
plot(x1,y)
for(i in 1:10){
  lines(xpred, bgibbs[i,1] + bgibbs[i,2]*xpred)
}
```



You can see within the loop that we're plotting our process model – b0 + b1*x – for pairs of regression parameters from the MCMC which created a distribution of models. The reason that we use pairs of values from the posterior (i.e. rows in bgibbs) rather than simulating values from the posterior of each parameter independently is to account for the covariance structure of the parameters. As we saw in the pairs plot above, the covariance of the slope and intercept is considerable, and thus independent sapling of their marginal posterior distributions would lead to credible intervals that are substantially too large. This distribution of models is by itself not easy to interpret, especially if we added lines for EVERY pair of points in our posterior, so instead we'll calculate the quantiles of the posterior distribution of all of these lines. To do this we'll need to first make the predictions for all of these lines and then look at the posterior distribution of predicted y values given our sequence of x values. Before we dive into that, lets also consider the other quantity we're interested in calculating, the predictive interval, since it is easiest to calculate both at the same time. While the credible interval was solely concerned about the uncertainty in the model parameters, the predictive interval is also concerned about the residual error between the model and the data. When we make a prediction with a model and want to evaluate how well the model matches data we obviously need to consider our data model. How we do this numerically is to generate pseudodata from our model, conditioned on the current value of not only the regression parameters but also the variance parameters, and then look at the distribution of predictions. In R we'll begin the calculation of our credible and predictive intervals by setting up data structures to store all the calculations

```
## credible and prediction intervals
xpred <- 0:20                   ## sequence of x values we're going to
npred <- length(xpred)              ##      make predictions for
ypred <- matrix(0.0,nrow=ngibbs,ncol=npred) ## storage for predictive interval
ycred <- matrix(0.0,nrow=ngibbs,ncol=npred) ## storage for credible interval
```

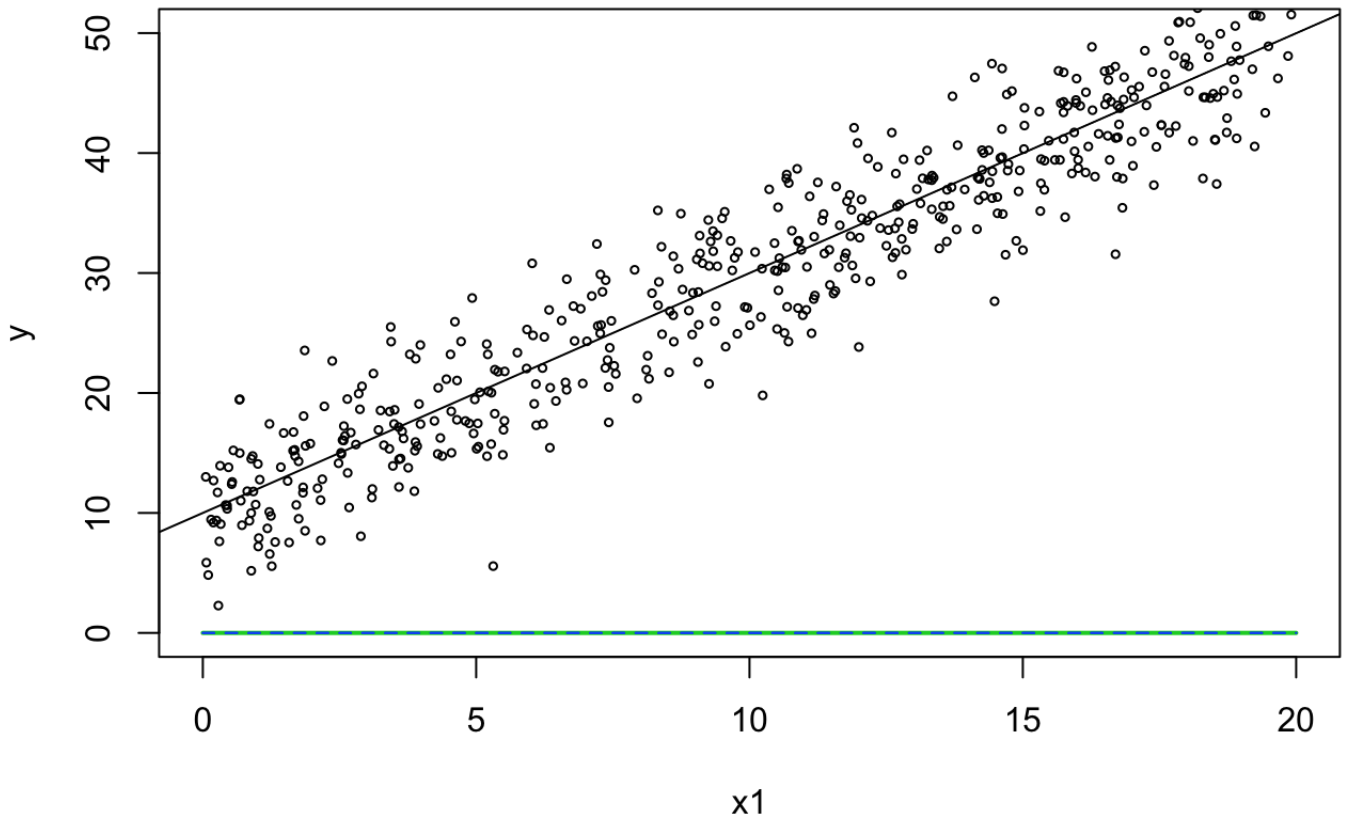
Next we'll set up a loop where we'll calculate the expected value of y at each x for each pair of regression parameters and then add additional random error from the data model. When looping through the posterior MCMC we'll obviously want to account for any burn-in period and thinning

```
for(g in seq(from=beg,to=ngibbs,by=thin)){
    Ey <- bgibbs[g,1] + bgibbs[g,2] * xpred
    ycred[g,] <- Ey
    ypred[g,] <- rnorm(npred,Ey,sqrt(sgibbs[g]))
}
```

Once we have the full matrix of predicted values we'll calculate the quantiles by column (ie for each x value) and then plot them vs. the data. By selecting the 2.5% and 97.5% quantiles we are generating a 95% interval, because 95% of the calculated values fall in the middle and 2.5% fall in each of the upper and lower tails. We could construct alternative interval estimates by just calculating different quantiles, for example the 5% and 95% quantiles would provide a 90% interval estimate.

```
ci <- apply(ycred,2,quantile,c(0.025,0.5,0.975))  ## credible interval and median
pi <- apply(ypred,2,quantile,c(0.025,0.975))        ## prediction interval

plot(x1,y,cex=0.5,xlim=c(0,20),ylim=c(0,50))
lines(xpred,ci[1,],col=3,lty=2) ## lower CI
lines(xpred,ci[2,],col=3,lwd=2) ## median
lines(xpred,ci[3,],col=3,lty=2) ## upper CI
lines(xpred,pi[1,],col=4,lty=2) ## lower PI
lines(xpred,pi[2,],col=4,lty=2) ## upper PI
abline(b0,b1)               ## true model
```



Lab Report Task 2: Power Analysis

The pseudodata we generated had a fairly large sample size (n=500) which allowed us to get a precise estimate of the true model. An important question in experimental design is what sample size is required in order to estimate model parameters accurately and to the required degree of precision. To take a rough first pass at this I'd like you to re-run this analysis two more times with a sample size of n=50 the first time and n=5 the second time. (note: I'll be asking you to make comparisons between the models so you'll want to take a look at what information you'll need to save from each run before re-running the model) For each run check the MCMC diagnostics and then report the following

- A summary table of parameter estimates
- A plot with the pseudo-data, the true model, the 95% credible interval on the model, and the 95% prediction interval. Overall also provide the following
- Plots showing how the estimate of the parameter mean and the 95% CI around that mean changes with sample size. Sample size should be on the x-axis on a log scale. Make plots for each of the 3 parameters
- Describe how sample size affects the ability to estimate the true model parameters. Does the 95% CI or PI ever fail to encompass the true model? How does the width of the CI change as a function of sample size?
- What is the (approximate) minimum sample size that would be required to reject the hypothesis that the slope of this line is 3/2 with 95% confidence.

Extra Credit: Lab Report Task 3: Informative Priors

Suppose a review of the scientific literature reveals that 95% of the reported values for the slope of the relationship between these two variables fall between 1.5 and 2.5. For the case where n=5 re-run your analysis with the same pseudodata as you did in Task 2 above but incorporating this additional information as a prior constraint on the slope. Include the following in your report

- A summary table of parameter estimates
- A plot with the pseudo-data, the true model, the 95% credible interval on the model, and the 95% prediction interval.
- Describe how the inclusion of prior information affects the estimate and CI for all of the parameters (not just the slope) and the overall CI and PI on the model