Exercise 05B - Bayesian Regression

```
## Loading required package: coda

## Linked to JAGS 4.3.0

## Loaded modules: basemod,bugs
```

Bayesian Regression using Gibbs Sampling

Linear regression is a good place to continue our exploration of Bayesian 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.

The standard Bayesian regression model assumes a Normal likelihood, a multivariate Normal prior on the vector of regression parameters, and an Inverse Gamma prior on the variance.

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

As a reminder, a Normal prior is not used on the variance because variances must be non-negative. The inverse gamma prior on the variance (equivalent to a gamma prior on the precision) has the advantage of being conjugate and having relatively interpretable parameters (s_1 and s_2 are proportional to the prior sample size and sum of squares, respectively; see Box 5.1)

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 model
  - specify parameters for the priors
  - specify initial conditions
- MCMC loop
- Code used to evaluate the analysis
  - Convergence diagnostics
  - Summary statistics
  - Credible & predictive intervals

Set up

Load data

Any data analysis begins with loading and organizing the data into a list so we can pass it to JAGS. However, for the first part of this activity 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 <- 100            ## 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
sigma <- 4              ## define the standard deviation
```

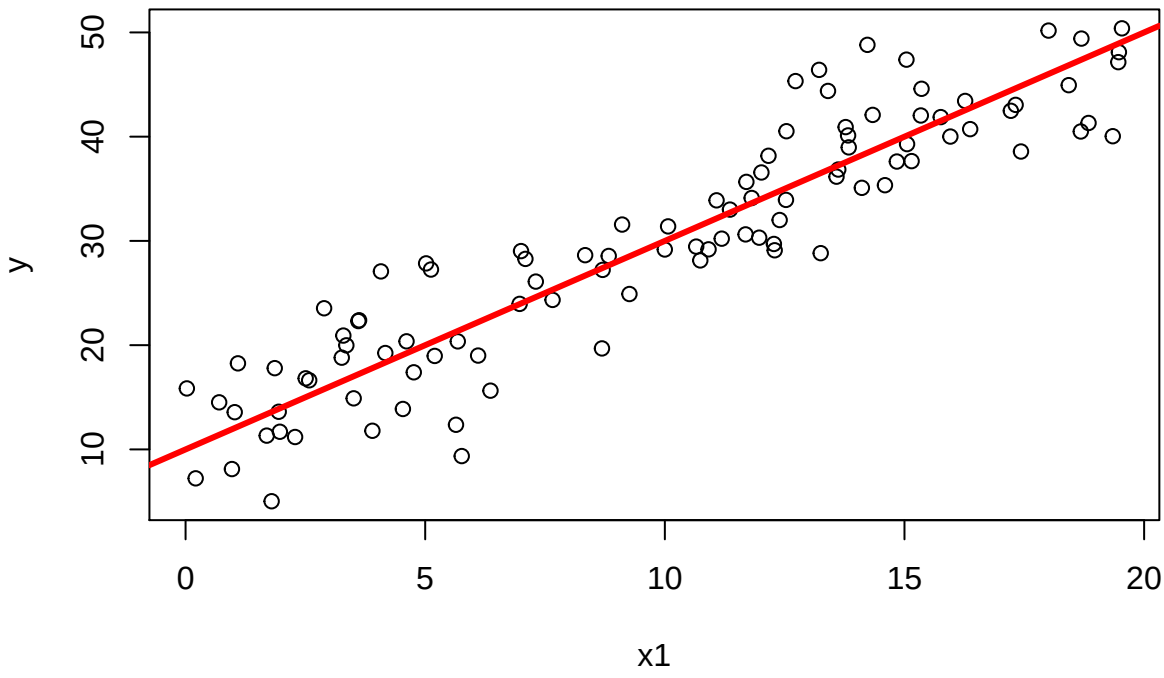
In this first part we define the true parameters of our model that we're going to simulate from. In the next block we will first simulate our covariate, x1, by assuming that it is uniformly distributed over the range that we're interested in (0,20) and then create our design matrix, x, which is a matrix with the first column of all 1's (which will multiplied by the intercept) and the second column being our covariate pseudodata (which will be multiplied by the slope). 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 we use matrix multiplication to multiply the design matrix, x, by the parameter vector, beta, the first column (all 1's) 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 <- rnorm(n,x%*%beta,sigma)
```

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.

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)
```



Finally, to be able to pass the data into JAGS we need to organize it into a list

```
data <- list(x = x1, y = y, n = n)
```

Specify model

Our JAGS model for regression is very similar to our model from Exercise 5. However, rather than having a single, fixed mu (expected value of y), we now index mu to vary by observation. We then add a linear process model to predict how mu changes as a function of x. Finally, we add a multivariate normal prior on the regression parameters (beta).

```
univariate_regression <- "
model{

  beta ~ dmnorm(b0,Vb)      ## multivariate Normal prior on vector of regression params
  prec ~ dgamma(s1,s2)    ## prior precision

  for(i in 1:n){
      mu[i] <- beta[1] + beta[2]*x[i]       ## process model
      y[i]  ~ dnorm(mu[i],prec)             ## data model
  }
}
"
```

Specify priors

Now that we have "data" and the model, the next task in setting up the analysis is to specify the parameters for the prior distributions.

Since beta is a vector, the prior mean, b0, 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 10000 (s.d. of 100). We'll use the *diag* command to set up a diagonal matrix that has a size of 2 and values of 10000 along the diagonal. In practice, JAGS requires that we specify the multivariate normal in terms of a precision matrix, therefore we will compute the inverse of the prior variance matrix, vinvert, using the *solve* function. For the residual error, we will specify an uninformative gamma prior on the precision with parameters s1 = 0.1 and s2 = 0.1. Finally, we'l add all of these prior parameters to the `data` object so we can pass them in to the JAGS model

```
## specify priors
data$b0 <- as.vector(c(0,0))      ## regression beta means
data$Vb <- solve(diag(10000,2))   ## regression beta precisions
data$s1 <- 0.1                    ## error prior n/2
data$s2 <- 0.1                    ## error prior SS/2
```

Note: If you want to fit a regression with more covariates (and thus more betas), you will need to increase the *size* of `b0` and `Vb` to match the number of betas.

Specify initial conditions

The last thing we need to do to prep for the MCMC loop is to specify the **initial conditions**. As with Exercise 5, we'll want to put these in a list. Here we're specifying multiple initial conditions and creating a list of lists so that we can give different initial conditions to each chain.

```
## initial conditions
nchain = 3
inits <- list()
for(i in 1:nchain){
 inits[[i]] <- list(beta = rnorm(2,0,5), prec = runif(1,1/100,1/20))
}
```

MCMC loop

As in Exercise 5, we'll pass the model, data, and inits to the `jags.model` function in order to compile and initialize the model

```
j.model   <- jags.model(file = textConnection(univariate_regression),
                        data = data,
                        inits = inits,
                        n.chains = nchain)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

```
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 2
##    Total graph size: 415
##
## Initializing model
```

And then we'll use the coda.samples function to sample from the posterior

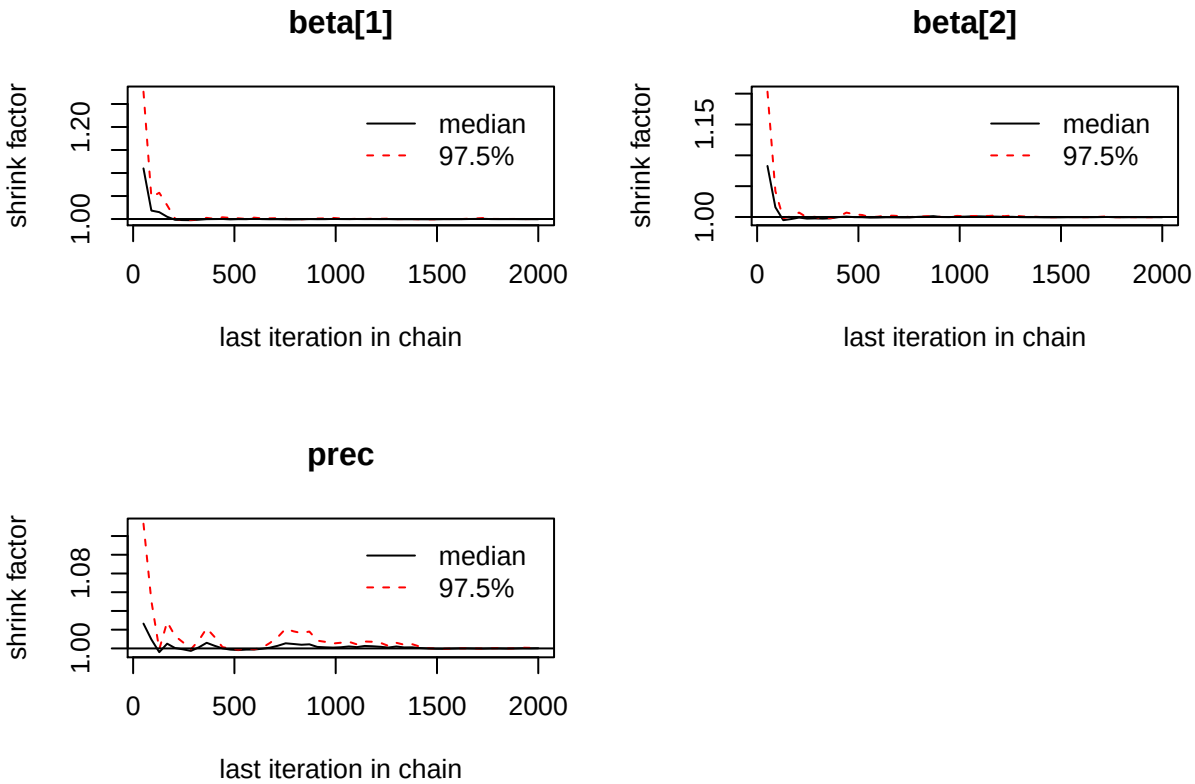
```
var.out   <- coda.samples (model = j.model,
                           variable.names = c("beta","prec"),
                               n.iter = 2000)
```

Evaluation

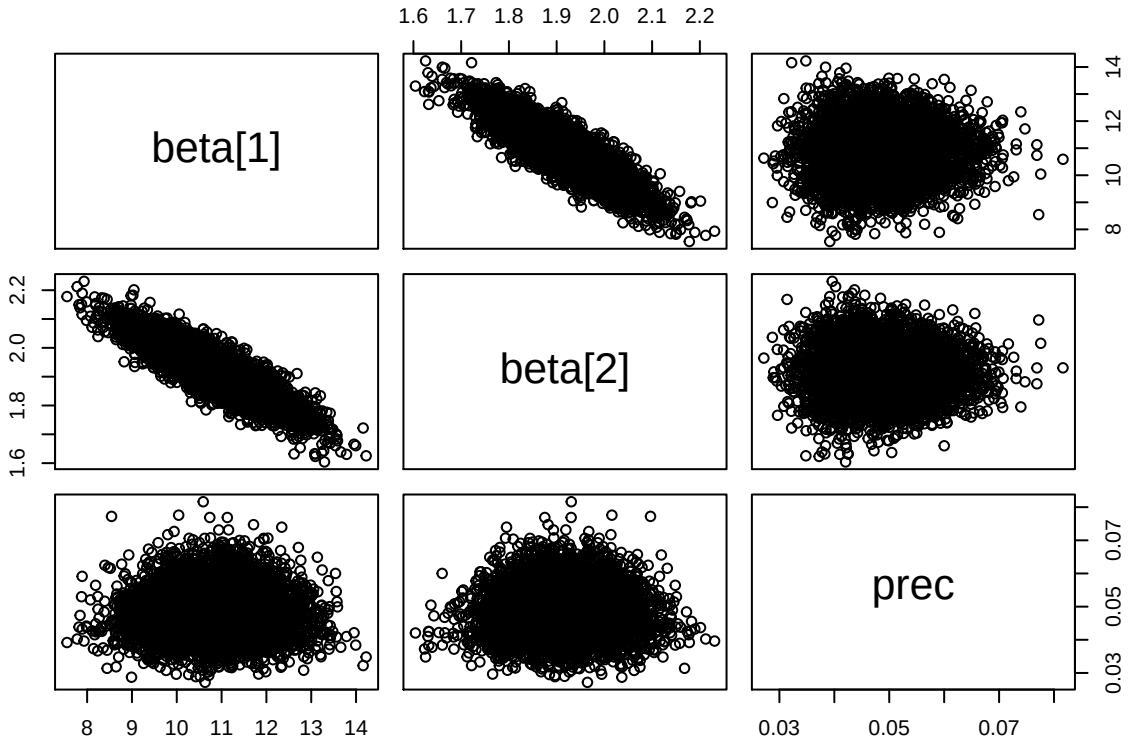
As we did in Exercise 5, we need to evaluate the output of the MCMC and will use the "coda" library to simplify these graphs and calculations.

There a few additional diagnostics we'll do that are not in coda, such as looking at parameter correlations, and to do those we'll want to convert the coda object to a matrix for easier manipulation. We'll also use this matrix later when creating the CI and PI around the regression.

```
## remember to assess convergence and remove burn-in before doing other diagnostics
GBR <- gelman.plot(var.out)
```



```
## convert to matrix
var.mat      <- as.matrix(var.out)

## Pairwise scatter plots & correlation
pairs(var.mat)  ## pairs plot to evaluate parameter correlation
```




```
cor(var.mat)    ## correlation matrix among model parameters
```

```
##              beta[1]      beta[2]         prec
## beta[1]  1.000000000 -0.870868518 -0.002597286
## beta[2] -0.870868518  1.000000000  0.001190165
## prec    -0.002597286  0.001190165  1.000000000
```

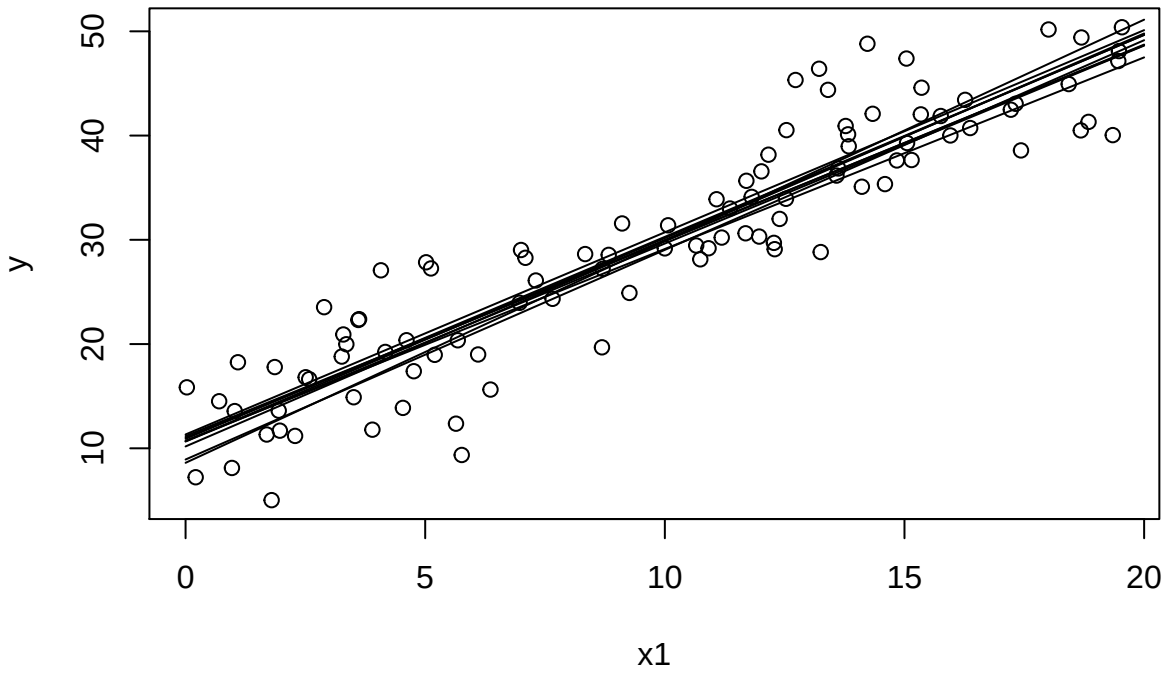
Task 1

- Evaluate the MCMC chain for convergence. Include relevant diagnostics and plots. Determine and remove burnin e.g. `var.burn <- window(var.out,start=burnin)`
- Report parameter summary table and plot marginal distributions
- Describe and explain the parameter covariances that you observe in the pairs plot and parameter correlation matrix.
- 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, var.mat[i,"beta[1]"] + var.mat[i,"beta[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

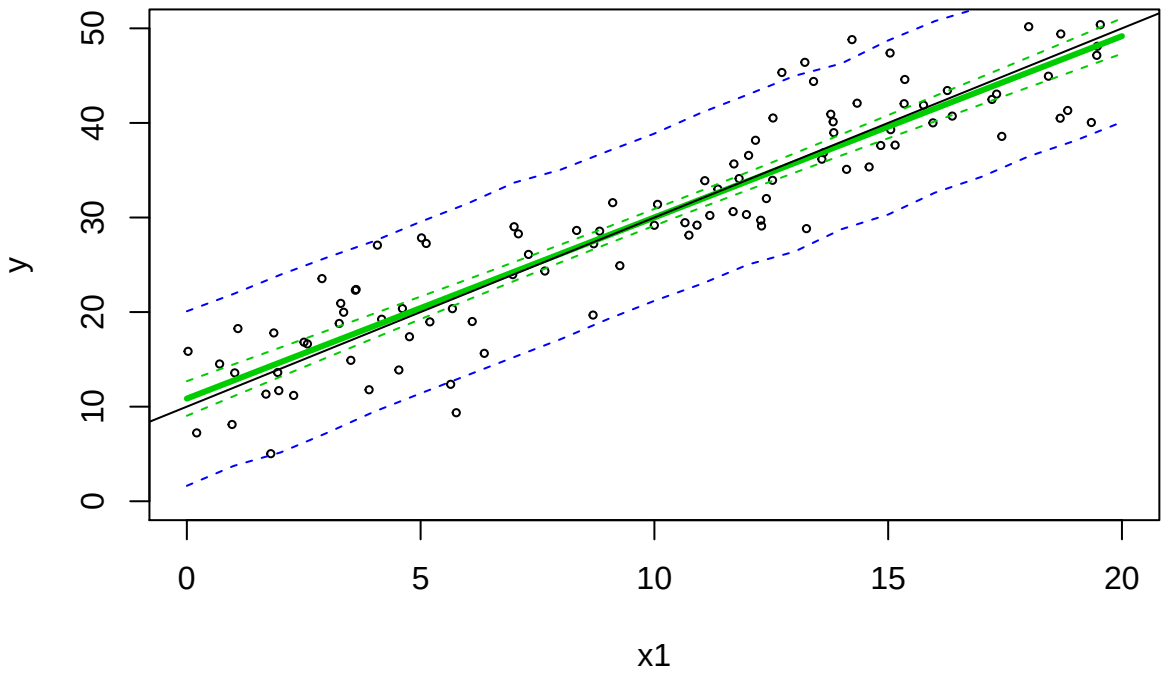
```
## credible and prediction intervals
nsamp <- 5000
samp <- sample.int(nrow(var.mat),nsamp)
xpred <- 0:20                    ## sequence of x values we're going to
npred <- length(xpred)               ##      make predictions for
ypred <- matrix(0.0,nrow=nsamp,ncol=npred)  ## storage for predictive interval
ycred <- matrix(0.0,nrow=nsamp,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_len(nsamp)){
  theta = var.mat[samp[g],]
  ycred[g,] <- theta["beta[1]"] + theta["beta[2]"]*xpred
  ypred[g,] <- rnorm(npred,ycred[g,],1/sqrt(theta["prec"]))
}
```

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.

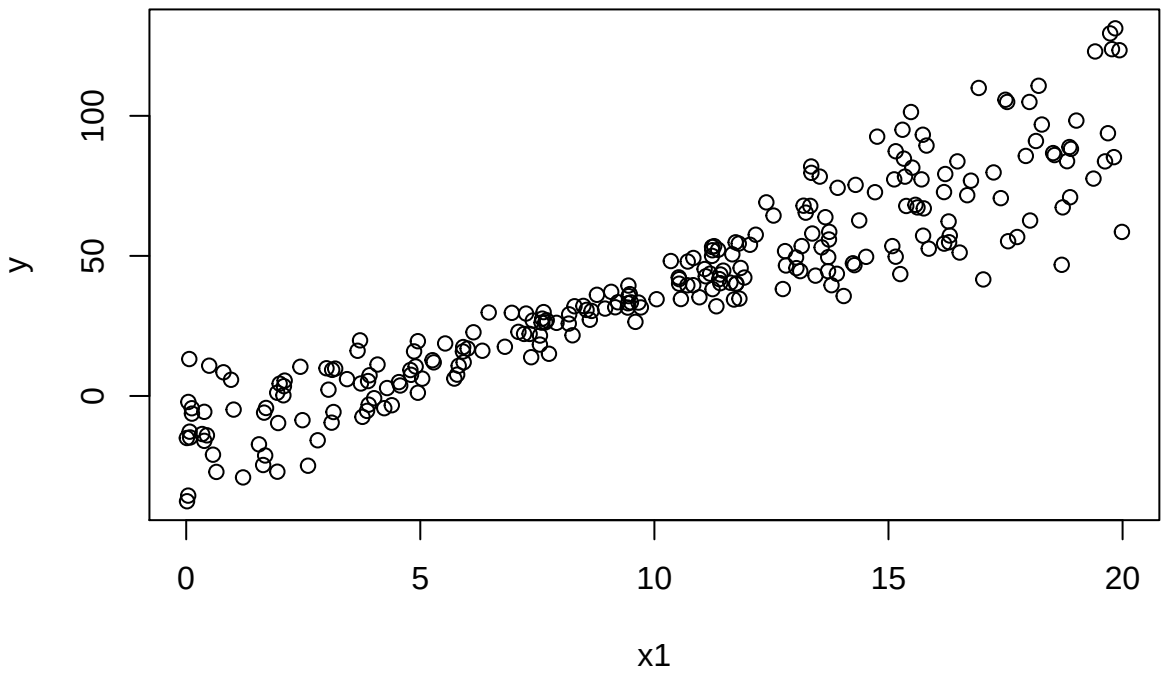
```
ci <- apply(ycred,2,quantile,c(0.025,0.5,0.975))  ## credible interval and median
pi <- apply(ypred,2,quantile,c(0.025,0.975))        ## prediction interval

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=3) ## 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
```



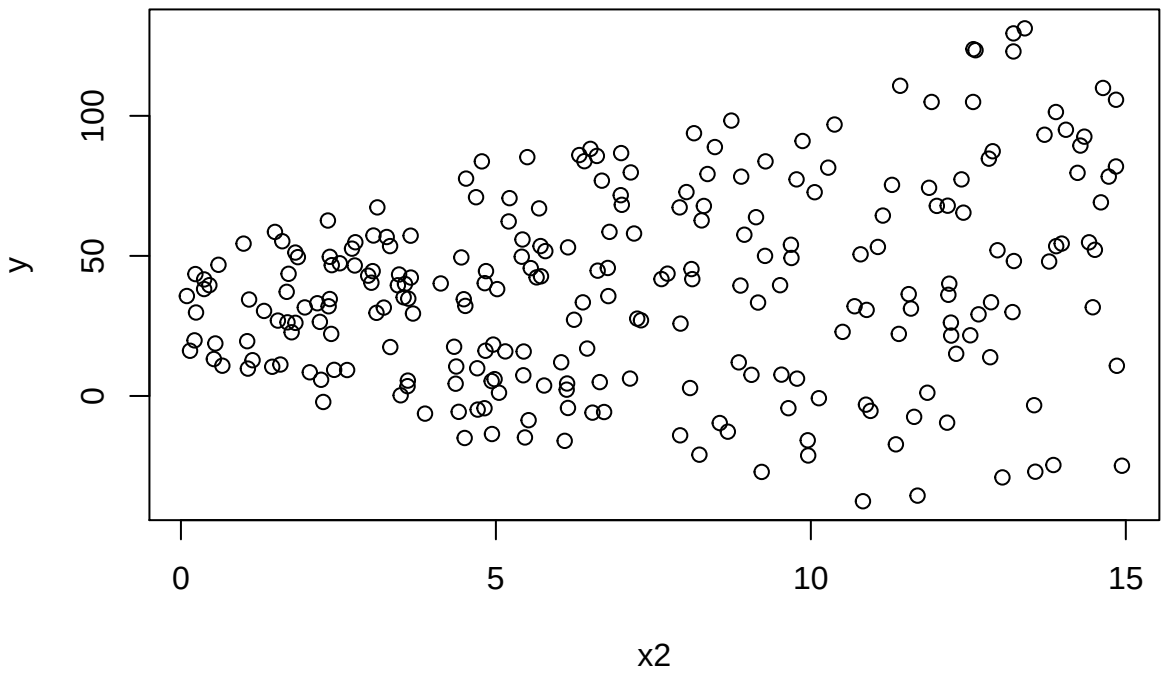
Multiple regression.

When building more complex models, it is often useful to start from a simple JAGS model and modify it to incrementally to add complexity. Here we're going to fit a multiple regression model with two covariates and an interaction term. For the following pseudodata:

```
### Part 1: simulate data from a known model
n <- 250            ## define the sample size
b0 <- 10                ## define the intercept
b1 <- 2                 ## define slope1
b2 <- -4        ## define slope2
b3 <- 0.5       ## define interaction
beta <- matrix(c(b0,b1,b2,b3),4,1)      ## put "true" regression parameters in a matrix
sigma <- 4              ## define the standard deviation
x1 <- runif(n,0,20)
x2 <- runif(n,0,15)
x <- cbind(rep(1,n),x1,x2,x1*x2)
y <- rnorm(n,x%*%beta,sigma)
plot(x1,y)
```




```
plot(x2,y)
```



Extend your univariate regression model to a multiple regression.

Hints: Make sure the length/dimensions of your priors and initial conditions match the number of parameters in your new process model

Task 2

- Show the JAGS and R code used.
- Include relevant convergence diagnostics and plots.
- Report parameter summary table.
- Plot marginal and pairwise joint distributions. Indicate 'true' parameters on the plots
- 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?