R Demonstration – Simple Linear Regression

Objective: The purpose of this week's session is to demonstrate how to perform simple linear regression (i.e. linear regression with a single predictor variable) in R and OpenBUGS. Regression diagnostics, such as plotting residuals, will be discussed, along with a simple logarithmic transformation to improve the linearity of the data. Finally, confidence intervals will be generated for the intercept and slope parameter estimates of the regression model, as well as for the fitted values and the prediction intervals.

Download the species-area data from the Preston (1962) study as spreadsheet text file (`Regression_Data.txt`) from the course website and save it in your `PCB6466` folder. You may also wish to download the script `Regression_2013.R`.

Part I. Regression on the Untransformed Data

Start the R software. From the menu bar, select *File→Change dir…* and then browse to the `PCB6466` folder on the Desktop. Enter the following commands to load and attach the Preston (1962) species-area data:

```
orig_data <- read.table("Regression_Data.txt", header=T)
attach(orig_data)
```

Now we will use the *`plot`* function to generate a scatterplot of the island area on the X-axis versus the number of plant species on the Y-axis:

```
plot(Area, Nspecies)
```

This relationship looks somewhat linear (though from the species-area relationship in ecology we know it should probably be exponential—see pages 224-225 of the Gotelli & Ellison text). We will thus use the *`lm`* function to define a linear model relating the number of species (`Nspecies`) to the area of the island (`Area`) and then store the result in a variable named `orig_model`:

```
orig_model <- lm(Nspecies ~ Area)
```

Next, we will use the *`summary`* function to produce summary statistics for our simple linear regression and explore the output generated by this function:

```
summary(orig_model)
```

```
Call:
lm(formula = Nspecies ~ Area)

Residuals:
   Min     1Q Median     3Q    Max
-88.55 -53.66 -43.56  21.21 216.04
```

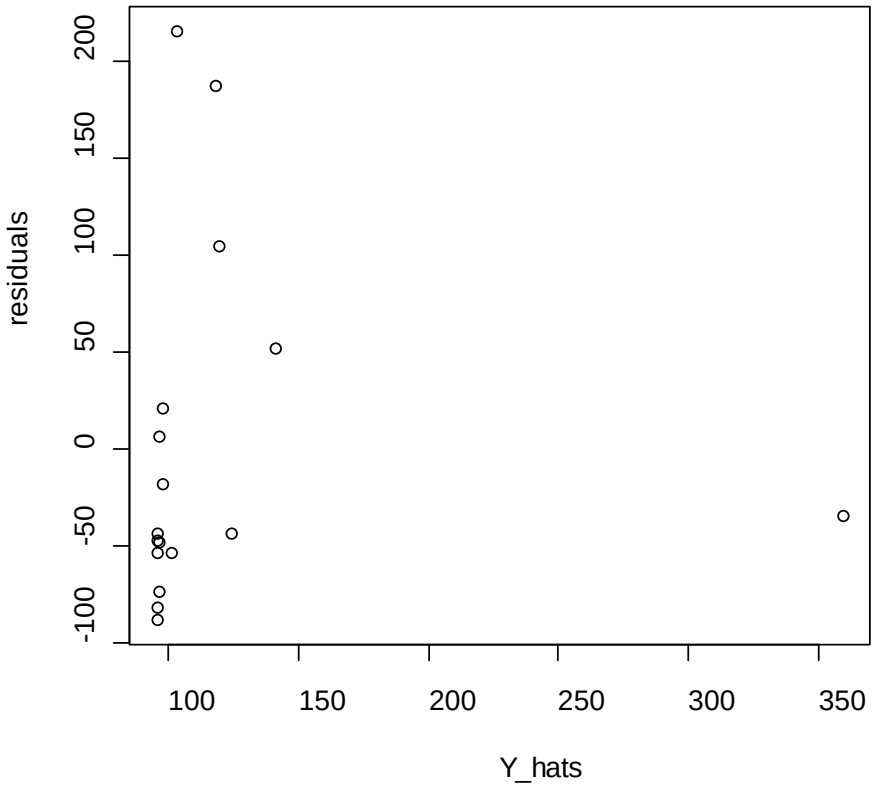
```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 95.44410   24.38400   3.914  0.00138 **
Area         0.04534    0.01677   2.703  0.01636 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 93.72 on 15 degrees of freedom
Multiple R-squared: 0.3275,     Adjusted R-squared: 0.2827
F-statistic: 7.306 on 1 and 15 DF,  p-value: 0.01636
```

There are several interesting things to note about the output of the *summary* function. First, the section labeled `Coefficients` contains information about the regression parameter estimates. The first row is the intercept coefficient (i.e., $\hat{\beta}_0$ in the regression model presented in class) and the second row is the slope coefficient ($\hat{\beta}_1$ in our regression model). In addition to the parameter **estimate** and its **standard error** (which we will use later to generate 95% confidence intervals for the parameters), notice that each coefficient has **a t-value** and **P-value** listed in the 3rd and 4th columns. The t-values are based on two-tailed tests of the null hypotheses that the coefficients are not statistically different from zero at the 5% significance level. Because the P-values for both coefficients are less than 0.05, we can conclude that they are both significant.

The `Residual standard error` is what we call the **mean square residual** (MS_{resid}), and is an estimate of the variance of the residual (or random) error from the regression. The next line provides two different estimates of the coefficient of determination ($\mathbf{R^2}$). We will discuss these more next week when we cover the topic of Multiple Regression. Finally, the last line provides the **F-ratio** and **P-value** for the overall regression. Again, these indicate that our simple linear regression of `Nspecies` versus `Area` is statistically significant.

Since our intercept and slope coefficient estimates are statistically significant and our overall regression is statistically significant, we might feel pretty confident about our results. But, as mentioned in the lecture and the Gotelli & Ellison text, one of the most important diagnostic tests for a linear regression is to plot the residuals against the fitted values to check the validity of the homogeneity of variance assumption. We use the *fitted* and *resid* functions to obtain these values and then draw a scatterplot:

```
Y_hats <- fitted(orig_model)
residuals <- resid(orig_model)
plot(Y_hats, residuals)
```



As shown in the scatterplot above, the variance is *heteroscedastic* (see page 260 of Gotelli & Ellison). In this particular case, the residuals appear to decrease with the fitted values. Thus, this regression of our untransformed values also appears to violate one of the fundamental assumptions: that the variances are constant along the regression line.

Part II. Regression on Log-Transformed Data

Given this apparent violation and our knowledge that the species-area relationship is usually exponential in nature, we can perform a transformation on both our independent and dependent variables to improve the linearity of their relationship. Specifically, we will take logarithms of each variable using the *log* function as shown here:

```
log_Area <- log(Area)
log_Nspecies <- log(Nspecies)
plot(log_Area, log_Nspecies)
```

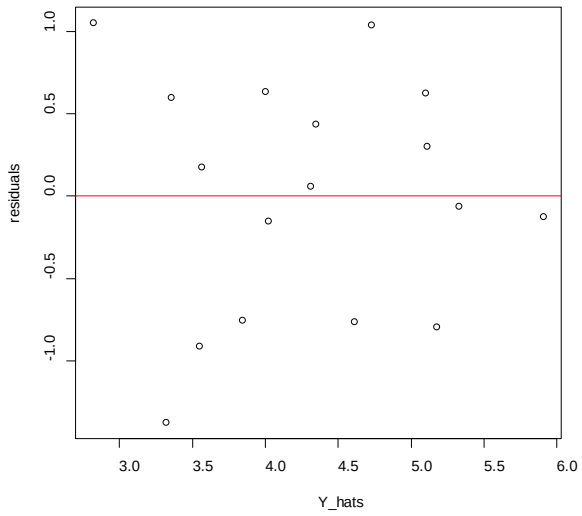
The scatterplot created with the *plot* function shows an apparent improvement in the linear relationship between the log-transformed number of species and the log-transformed area. Now we can create a new linear model (`trans_model`) based on the log-transformed variables and then perform the simple linear regression as before:

```
trans_model <- lm(log_Nspecies ~ log_Area)
summary(trans_model)
```

```
Call:
lm(formula = log_Nspecies ~ log_Area)

Residuals:
     Min       1Q   Median       3Q      Max
-1.37282 -0.75233  0.06034  0.59768  1.04971

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.03895    0.32728   9.286 1.31e-07 ***
log_Area     0.33059    0.07194   4.595  0.00035 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7378 on 15 degrees of freedom
Multiple R-squared: 0.5847,     Adjusted R-squared: 0.557
F-statistic: 21.12 on 1 and 15 DF,  p-value: 0.0003501
```

As was the case with the untransformed regression, the output indicates that our results are statistically significant. The P-values for the null hypothesis of the slope and intercept coefficients and the P-value for the overall regression are all highly significant (i.e., $P < 0.001$). Furthermore, the R^2 values for the log-transformed regression are almost twice those for the untransformed model, suggesting that the log-transformed model explains about two times the amount of the variation in our response variable. Finally, we can once again plot the residuals against the fitted values to check that the assumption of homogeneity of variance along the regression line is true:

```
Y_hats <- fitted(trans_model)
residuals <- resid(trans_model)
plot(Y_hats, residuals)
abline(h=0, col="red")
```



As the scatterplot on the previous page shows, the residuals appear to neither increase nor decrease in value in any systematic manner as we move along the X-axis. Thus, our log-transformed regression appears to meet the homogeneity of variance assumption whereas our untransformed regression model did not.

Part III. Confidence Intervals

As discussed on pages 253-257 of the Gotelli & Ellison text, confidence intervals can be created for the intercept and slope parameters of a simple linear regression. To create these intervals, we will need to obtain estimates of the standard error for each of these parameter estimates. Fortunately, the *summary* function provides a way to easily obtain the standard error estimates for the coefficients:

```
coeffs <- summary(trans_model)$coefficients
```

This command retrieves the coefficients as a matrix from our transformed regression model and stores them in a variable named `coeffs`. As shown below, the first row of the matrix contains the intercept coefficient and the second row contains the slope coefficient. Each row contains the parameter and standard error estimates in the 1st and 2nd columns, respectively:

```
coeffs
             Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 3.0389536 0.32727773 9.285550 1.313053e-07
log_Area    0.3305944 0.07193987 4.595427 3.501432e-04
```

Next, we need the critical value from the t-distribution with n-2 degrees of freedom, which we can calculate with the following command:

```
t_crit <- qt(0.975, length(log_Area)-2)
```

Now we have all the data we need to construct the 95% confidence intervals for our regression parameter estimates. The following block of code creates and displays the 95% confidence interval for β_0, the **intercept** parameter in our regression model:

```
B0_hat <- coeffs[1,1]
B0_se <- coeffs[1,2]
B0_lower <- B0_hat - (B0_se * t_crit)
B0_upper <- B0_hat + (B0_se * t_crit)
cat("95% CI for B0:[", B0_lower,",",B0_upper,"]\n",sep="")
95% CI for B0: [2.341378, 3.73653]
```

Similarly, the following code creates the 95% confidence interval for β_1, the **slope** parameter in our regression model:

```
B1_hat <- coeffs[2,1]
B1_se <- coeffs[2,2]
B1_lower <- B1_hat - (B1_se * t_crit)
B1_upper <- B1_hat + (B1_se * t_crit)
cat("95% CI for B1:[",B1_lower,",",B1_upper, "]\n", sep="")
95% CI for B1: [0.1772582, 0.4839307]
```

We can also generate 95% confidence intervals for both the fitted values and the predicted values based on our regression model. As noted on pages 255-257 of the Gotelli & Ellison text, there is a subtle but important distinction between the confidence intervals for the fitted values (i.e., the $\hat{Y}$s) and those for predicted values (so-called *prediction intervals*). The confidence intervals for the fitted values will flare out in either direction as you move away from the mean of the predictor variable. Thus, confidence intervals for fitted values will always be smaller for interpolation than for extrapolation. On the other hand, the prediction intervals will always be wider than the fitted value confidence intervals because they incorporate greater uncertainty about the predicted value of the response variable.

To generate our confidence intervals for the fitted values and the prediction intervals, we will first create a new data frame object to hold these intervals:

```
pred_frame <- data.frame(log_Area)
```

Our new data frame, `pred_frame`, starts with one row for each of the islands and a single column containing the log-transformed area of each island. Next, we use the *`predict`* function to generate the confidence intervals for the fitted values and the prediction intervals:

```
conf_intervals<-predict(trans_model,int="c",newdata=pred_frame)
pred_intervals<-predict(trans_model,int="p",newdata=pred_frame)
```

The first call to the *`predict`* function uses the `int="c"` argument to indicate that *confidence intervals* for the fitted values should be calculated and stored in the variable `conf_intervals`. The subsequent call to the *`predict`* function uses the `int="p"` argument to indicate that *prediction intervals* should be calculated and stored in the variable `pred_intervals`. Both of these calls will produce 95% confidence intervals, since that is the default for the *`predict`* function and we have not specified otherwise.

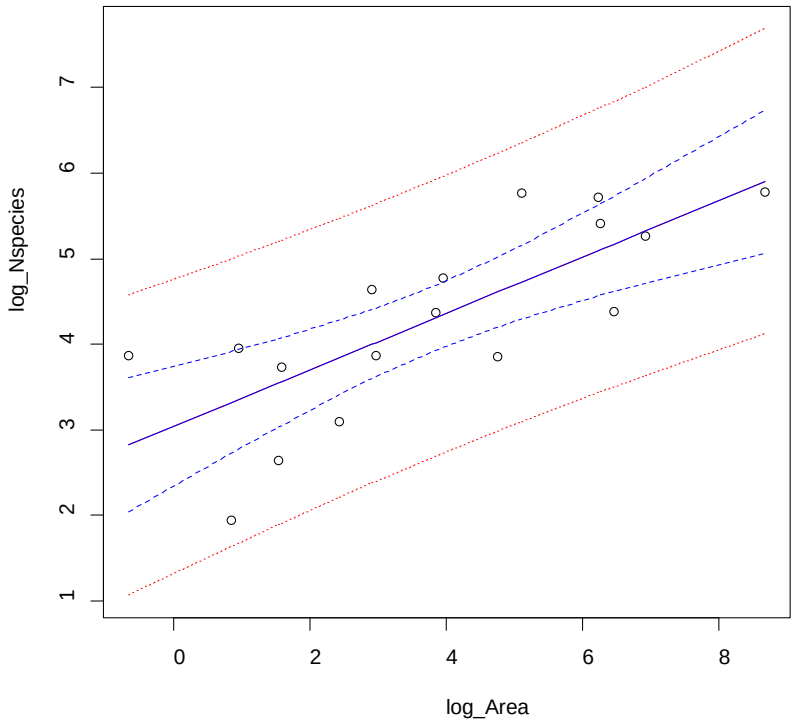
Now we are finally ready to plot all of our results. First, we will use the familiar *`plot`* function to draw a scatterplot of the log-transformed number of species versus the log-transformed island areas:

```
plot(log_Area,log_Nspecies,ylim=range(log_Nspecies,pred_intervals))
```

Next, we will use the *`matlines`* function to draw the fitted regression line, the fitted value confidence intervals, and the prediction intervals. The *`matlines`* function allows us to plot multiple lines for the same points on the X-axis, which is what we need to create the lines for our confidence intervals:

```
matlines(pred_frame$log_Area, pred_intervals, lty=c(1,3,3), col="red")
matlines(pred_frame$log_Area, conf_intervals, lty=c(1,2,2), col="blue")
```

The first call to the *matlines* function draws dotted red lines for the prediction intervals, and the second call draws dotted blue lines for the fitted value confidence intervals. The following figure shows the results of all these plotting commands. If you are still confused as to what all of the lines mean, please see Figure 9.4 on page 256 of the Gotelli & Ellison text (**notice that here we used natural logarithms**).



Part IV. Simple linear regression with a Bayesian approach

Next, we implement a Bayesian analysis of the same model. Once again, we use the function library (R2OpenBUGS) to call the program that connects with OpenBUGS, and write the OpenBUGS program as described below. We need to provide three sets of data, *n* the number of islands, the logarithm of their areas and the logarithm of the number of species per area. The following lines of code describe the model, the uninformed prior distributions, and give the initial conditions and MCMC simulation parameters. This allows us to calculate the predicted values and use them to plot the regression model.

```
library(R2OpenBUGS)
n <- NROW(regression_data)
x <- log_Area
y <-log_Nspecies

### Analysis using OpenBUGS
# Write model
linreg<-function(){
  for (i in 1:n)                    # for each of the sites
  {
```

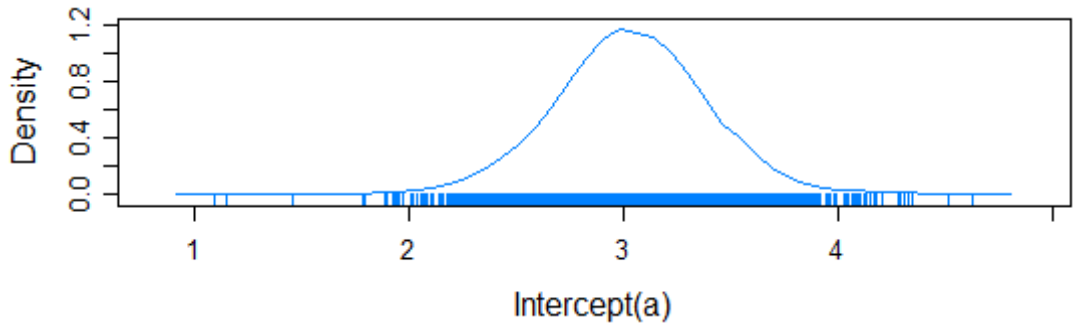
```
    y[i] ~ dnorm(mean[i], prec)     # assume normal distribution
    mean[i] <- a + b*x[i]
  }
  # uninformative priors
  a ~ dnorm(0, 1.0E-6)
  b ~ dnorm(0, 1.0E-6)
  prec ~ dgamma(0.001, 0.001)
  for (j in 1:10)
  {
    pred[j] <- a + b*(j)
  }
}
write.model(linreg, "linreg.txt")

# Bundle data
win.data <- list("x","y","n")
# Inits function
inits <- function(){ list(a=runif(1),b=runif(1),prec = 10)}
# Parameters to estimate
params <- c("a","b","prec","pred")
# MCMC settings
nc = 1
ni=40000
nb=400
nt=20
# Start Gibbs sampler
out <- bugs(data = win.data, inits = inits, parameters = params, model
= "linreg.txt",
n.thin = nt, n.chains = nc, n.burnin = nb, n.iter = ni, codaPkg=T)

library(coda)
reg.coda<-read.bugs(out)
```

We can then explore the numerical output and the accompanying plots for the distribution of the estimated parameters and their autocorrelation (remember with Bayesian analysis, your numerical results may be slightly different but should be commensurate with the frequentist analysis):

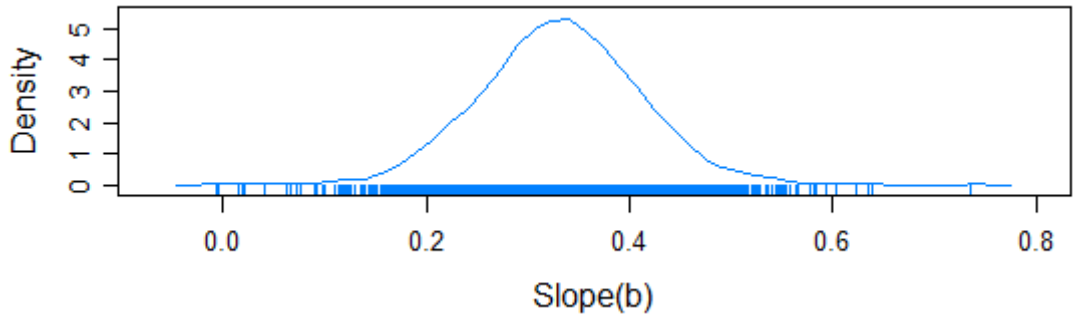
```
summary(reg.coda)
Iterations = 401:4000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 3600

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:
           Mean      SD Naive SE Time-series SE
a         3.0346 0.36053 0.006009       0.017466
b         0.3317 0.07906 0.001318       0.003838
deviance 39.1621 2.75421 0.045903       0.081712
prec      1.8152 0.67813 0.011302       0.013330
pred[1]   3.3663 0.29720 0.004953       0.013683
pred[2]   3.6980 0.24328 0.004055       0.009944
pred[3]   4.0297 0.20625 0.003438       0.006320
pred[4]   4.3613 0.19595 0.003266       0.003231
```

```
pred[5]   4.6930 0.21623 0.003604       0.003273
pred[6]   5.0247 0.26002 0.004334       0.006352
pred[7]   5.3564 0.31775 0.005296       0.009968
pred[8]   5.6880 0.38318 0.006386       0.013704
pred[9]   6.0197 0.45298 0.007550       0.017484
pred[10]  6.3514 0.52540 0.008757       0.021287

2. Quantiles for each variable:
             2.5%     25%     50%     75%   97.5%
a          2.3289  2.8050  3.0330  3.2613  3.7450
b          0.1808  0.2813  0.3324  0.3825  0.4892
deviance 36.0097 37.1475 38.4500 40.4625 46.2302
prec       0.7760  1.3310  1.7200  2.1963  3.4150
pred[1]    2.7870  3.1738  3.3660  3.5542  3.9450
pred[2]    3.2180  3.5410  3.6950  3.8550  4.1871
pred[3]    3.6290  3.8930  4.0270  4.1620  4.4410
pred[4]    3.9670  4.2300  4.3620  4.4830  4.7520
pred[5]    4.2670  4.5520  4.6930  4.8270  5.1230
pred[6]    4.5140  4.8580  5.0240  5.1950  5.5430
pred[7]    4.7410  5.1530  5.3570  5.5580  5.9830
pred[8]    4.9300  5.4390  5.6910  5.9320  6.4360
pred[9]    5.1258  5.7280  6.0210  6.3130  6.9091
pred[10]   5.3199  6.0140  6.3545  6.6960  7.3850
```

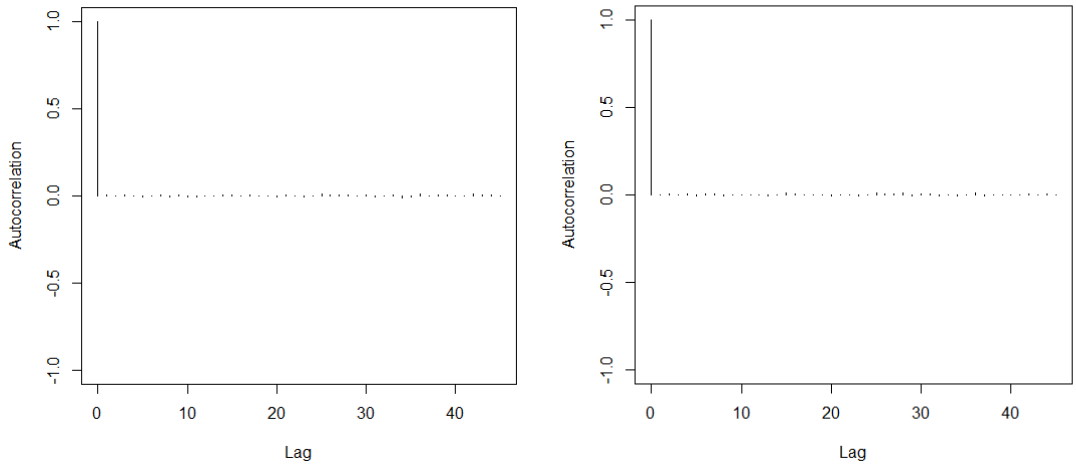
```
densityplot(reg.coda[,1], xlab="Intercept(a)")
```




```
densityplot(reg.coda[,1], xlab="Slope(b)")
```

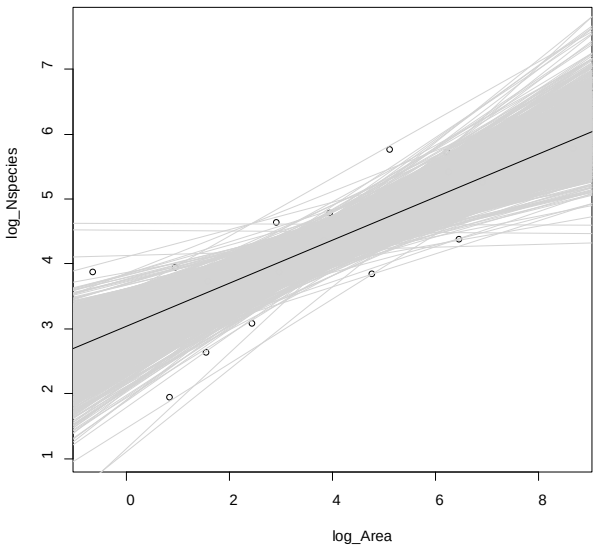


```
autocorr.plot(reg.coda[,1])
autocorr.plot(reg.coda[,2])
```



Finally, we can plot the Bayesian model and its credibility envelope together with the observed values, so that we can make sure that our results are similar to the ones we obtained with the frequentist model.

```
##Plot the fitted model together with the credibility envelope
and the observed values

results<-summary(reg.coda)plot(log_Area, log_Nspecies,
ylim=range(log_Nspecies, pred_intervals))
for(i in 1:1000){
abline(as.numeric(reg.coda[i,1]),as.numeric(reg.coda[i,2]),col="l
ightgray")}
abline(results$statistics[1,1],results$statistics[2,1])
```



The plot above shows the mean values of the Bayesian model (black line), the observed values (dots) and the fitted lines for 1,000 iterations (grey lines). Comparing plots we can see that the results of both analyses are remarkably similar. A comparison of the regression parameters and their confidence and Bayesian credible intervals should emphasize this point.

95 % Confidence Intervals (frequentist)

	mean	sd	2.5%	97.5%
Intercept	3.0	0.4	2.3	3.7
Slope	0.3	0.1	0.2	0.5

95 % Credibility Intervals (Bayesian)

	mean	sd	2.5%	97.5%
Intercept	3.0	0.4	2.3	3.7
Slope	0.3	0.1	0.2	0.5

Remember that the interpretation of the confidence interval is that if we take 100 replicated sampled observations of the same islands, in the same conditions, and we estimate one hundred 95% CI of the slope and intercept using linear regression, then on average we would expect 95 intervals would indeed contain the “true value” of these parameters. However, we cannot make any statement on the probability associated with these estimates.

On the other hand, the posterior probability in the Bayesian analysis measures our degree of belief in the estimated parameters, given the model, the observed data and our priors (which in this case were uninformative). From the Bayesian analysis we can make direct probability statements on the magnitude of the estimated parameters with the posterior probability. For example, we can state that the values indicate a clear increase in the number of species with island area because the probability that the slope is less than or equal to 0 is extremely small (and far outside of the 95% credibility interval). This is a more direct and satisfactory statement from the analysis.

As always, we finish our session by detaching the data we attached at the beginning of our R session:

```
> detach(orig_data)
```