Analysis of Environmental Data

Resampling Procedures

(Written by Kevin McGarigal, but borrowed heavily from Michael Crawley, The R Book (2007))

The purpose of this lab exercise is to familiarize you with using R to resample data for various purposes, focusing on the bootstrap and randomization tests. Resampling can be a powerful means of getting the most out of your data when classical statistical procedures fail or are inappropriate. It is assumed that you are already familiar with the classical statistical tests, so the focus in this lab will be on the practical aspects of how to use R to do resampling. Here is an outline of what is included in this lab exercise:

1. Set up your R work session. 1
2. Bootstrap. 1
   2.1 Boostrap confidence interval. 2
   2.2 A more complex example – rarefaction curves 4
3. Randomization Tests. 6
   3.1 A simple example – linear regression. 6
   3.2 A more complex example -- mantel test. 9
4. Exercise. 12

1. Set up your R work session

Open R and set the current working directory to your local workspace, for example:

```
setwd('c:/work/stats/ecodata/lab/resampling/')
```

Load the biostats library (which is not a formal library) by typing, substituting the appropriate path:

```
source('.../biostats.R')
```

2. Bootstrap

You have probably heard the old saying about 'pulling yourself up by your own bootlaces'. That is where the term 'bootstrap' comes from. It is use in the sense of getting 'something for nothing'. The idea is very simple. You have a single sample of n measurements, but you can sample from this in very many ways, so long as you allow some values to appear more than once, and other samples to be left out (i.e., sampling with replacement). All you do is draw observations at random from the original sample, replacing each observation after it is selected so that it has the same chance of being drawn on the next draw. By doing this repeatedly, you can create a new data set by resampling the original data set.

The bootstrap has many applications in statistics, but by far its most important use involves calculating non-parametric confidence intervals for parameter estimates. In this context, the bootstrap simulates the frequentist concept of obtaining estimates from repeated similar experiments. It substitutes resampling of one data set for repeated experiments. In this section, we will illustrate the use of bootstrap to calculate a nonparametric confidence interval on a simple

parameter estimate and then in the following section illustrate a more complex use of the bootstrap.

2.1 Boostrap confidence interval

Let's begin by reading in some data. The example data represent standardized abundance estimates of 10 rare moth species across 24 sample plots in the pine barrens of southeast Massachusetts.

```
moths<-read.csv('moths.csv',header=TRUE)
moths
```

For now, let's focus on just one species, the spiny oakworm (*Anisota stigma*), abbreviated 'anst', and examine its distribution with a histogram:

```
hist(moths$anst)
```

The first thing you should notice is that the distribution is grossly non-normal. Let's say we want to obtain a 95% confidence interval for the mean. The usual way to do this is to calculate the mean and assume the sampling distribution for the mean is normally distributed, which is essentially true under the central limit theorem as long as the sample size is large. In this case, we can use the student's t-distribution (which is essentially a normal distribution when the population variance is unknown and thus must be estimated from the sample, and has slightly heavier tails than the normal and approaches a normal distribution for large sample sizes greater than about 100) to obtain the critical value of t that corresponds to a tail probability of 2.5% (since we have two tails that must add up to 5%, leaving the central 95% for our confidence interval). Once we have the critical t value, we multiply it by the computed standard error of the mean (given by the standard deviation divided by the square root of the sample size) to determine the length of one side of the interval in the original units of measurement. Lastly, we add and subtract the computed half interval to derive the lower and upper confidence limits, as follows:

```
anst<-moths$anst
(m.obs<-mean(anst))
ci.obs<-qt(.975,df=length(anst)-1)*sd(anst)/sqrt(length(anst))
m.obs-ci.obs
m.obs+ci.obs
```

Note, in the above formula we used the qt() function to get the critical t value for a ones-sided evaluation, with degrees of freedom = n-1, and then we multiplied this by the standard error of the mean.

As the sample size increases (>>100), the critical value of t approaches 1.96. Thus, it is not uncommon when sample sizes are large to simply multiply the standard error by 1.96 to get the half interval, as follows:

```
ci.obs<-1.96*sd(anst)/sqrt(length(anst))
m.obs-ci.obs
m.obs+ci.obs
```

This is the standard way to construct a confidence interval for the mean and test the hypothesis that the mean differs from 0. If the confidence interval does not contain 0, then we can say that the mean is unlikely to have come from a distribution with a mean of 0.

An alternative to the parametric approach above is to use bootstrapping to obtain a confidence interval for the mean, and this would likely be a more robust estimate if the sample size is small and the underlying distribution of the variable is nonnormal. We begin by creating an empty vector to hold the results of the bootstrap:

```
m<-10000
result<-numeric(m)
```

Then we sample with replacement from the data using the sample() function with replace=TRUE to ensure that sampling is done with replacement, place the result in the holding vector (result), and do this inside of a loop so that we can repeat the process over and over:

```
for(i in 1:m){
    result[i]<-mean(sample(anst,replace=TRUE))
}
```

The vector result now contains 10,000 bootstrap sample means. We can calculate the mean of the bootstrap means and, more importantly, the 2.5% and 97.5% quantiles of the bootstrap distribution, as follows:

```
mean(result)
quantile(result,c(0.025,0.975))
```

How does this compare with parametric confidence interval? Close, but not identical. Our bootstrap confidence intervals are skewed because the data are skewed, whereas the parametric confidence interval is symmetric.

Now let's see how to do the same thing using the boot() function from the library called boot. First, we must load the library (assuming that you have already installed it from the R project):

```
library(boot)
```

The syntax of boot is very simple:

```
boot(data,statistic,R)
```

The trick to using boot lies in understanding how to write the 'statistic' argument. 'R' is the number of resamplings you want to do (e.g., R=10000), and 'data' is the name of the data object to be resampled (anst in this case).

Here is our statistic function:

```
mymean<-function(anst,i)  mean(anst[i])
```

This warrants some explanation. The attribute we want to estimate repeatedly is the mean value of 'anst'. Thus, the first argument to our statistic function must be 'anst'. The second argument is an index (a vector of subscripts) that is used within boot to select random assortments of anst. Our statistic function uses the built-in function mean() to calculate the mean value of the sample values. The key point is that we write mean(anst[i]) not mean(anst). Now we can find the bootstrap for 10000 iterations:

```
myboot<-boot(anst,mymean,R=10000)
myboot
```

The output is interpreted as follows. The 'original' is the original mean of the whole sample:

```
mean(anst)
```

while 'bias' is the difference between the original mean and the mean of the bootstrapped samples which are in the variable called myboot$t:

```
mean(myboot$t)-mean(anst)
```

and 'std.error' is the standard deviation of the simulated values in myboot$t:

```
sd(myboot$t)
```

Lastly, we can extract our bootstrap confidence interval as follows:

```
quantile(myboot$t,c(0.025,0.975))
```

How does this compare to the one we calculated ourselves? They differ slightly because they were generated with different series of random numbers.

2.2 A more complex example – rarefaction curves

Now that you understand the basics behind using the bootstrap, let's try using the bootstrap in a more complex application involving computing rarefaction curves. The principle behind rarefaction is that the number of species detected is typically a function of sampling intensity – the greater the sampling intensity (e.g., number of sample plots or sampling area) the greater the species richness. This is just the well-known species-area relationship. When the sampling intensity varies among sampling units, rarefaction can be used to adjust the species richness estimates so that they are comparable. A rarefaction curve shows the species richness as a function of sampling intensity. Bootstrapping is the procedure used to generate the rarefaction curve.

Let's begin with our moth data set by first removing the first column which represents an arbitrary site id and is not useful here.

```
data<-moths[,-1]
data
```

Next, let's create some objects to hold some quantities to make the subsequent script more compact:

```
n<-nrow(data) #number of rows or sample observations
m<-10000 #number of bootstrap iterations
```

Next, we need to create a data matrix to hold the bootstrap results. In this case, we will need a row for each bootstrap iteration and a column for each sampling intensity – which can range from a single observation to the full data set (n):

```
result<-matrix(nrow=m,ncol=n)
```

Next, we need to use a few tricks. First, we need to draw a bootstrap sample from the data set of a specified size. We can do this using indexing and sample() for the row index of our data frame: data[sample(),]. This says to select the rows from data corresponding to the result of the sample() function. Inside the sample() function we need to specify a vector containing a list of numbers ranging from 1 to n, the size of the bootstrap sample (i.e., number of row observations to take), and the replace=TRUE to ensure sampling with replacement. Once we have drawn a bootstrap sample, we compute the species richness of the sample. We can do this using the apply() function to sum by column (species) and then count how many species have sums>0 (indicating presence). We need to store the result in the appropriate location in the result matrix we created above. Finally, we need to put this whole set of operations into a double loop. The inside loop will create bootstrap samples of size 1 to n; the outside loop will iterate through 10,000 bootstrap iterations. This is what it looks like all together:

```
for(i in 1:m){
    for(j in 1:n){
        t1<-data[sample(1:n,size=j,replace=TRUE),]
        t2<-apply(t1,2,sum)
        result[i,j]<-sum(t2>0)
    }
}
```

After you execute the script above, it make take several minutes to run, so be patient. When it is finished, the object result now contains 10,000 rows and 24 columns, where each row is a separate bootstrap iteration and each column represents the size of the bootstrap sample, in this case representing the number of sample observations or sampling intensity ranging from 1 to 24. We can calculate the mean and 2.5% and 97.5% quantiles of the bootstrapped species richness for each sampling intensity using the apply() function, as follows:

```
rare.mean<-apply(result,2,mean)
rare.quant<-apply(result,2,quantile,probs=c(0.025,0.975))
```

For convenience, let's bind the objects together and transpose the data frame so that the columns

represent the mean, 2.5% and 97.5% quantiles, and the rows represent sampling intensity ranging from 1 to 24.

```
rare<-t(rbind(rare.mean,rare.quant))
```

We can plot the rarefaction curve and the 95% confidence interval using the matplot() function, which is useful for simultaneous plotting of several columns of a data frame or matrix:

```
matplot(rare,type='l',xlab='Number of samples',ylab='Species richness', main='Rarefaction Curve')
```

Lastly, let's add a legend to make the plot complete, as follows:

```
legend('bottomright',legend=c('mean','2.5%','97.5%'),lty=c(1,2,3),col=c(1,2,3), inset=c(.1,.1))
```

What does the rarefaction curve indicate about the species-sampling intensity relationship? Do you think 5 sample plots are sufficient to get a reliable estimate of then number of species present in the study area? What about 10 or 15? If you sampled 10 plots, what is your estimate of the number of species present and what is your confidence in that estimate?

We're done. Now isn't that an interesting use of the bootstrap?

3. Randomization Tests

The bootstrap offers a powerful means of quantifying uncertainty in parameter estimates when parametric approaches are suspect and, as such, it can be used to construct hypothesis tests. For example, does the 95% bootstrap confidence interval contain zero? If not, we can reject the null hypothesis that the parameter is equal to zero with high confidence. An alternative to the bootstrap for testing null hypotheses is the Monte Carlo randomization test. Like the bootstrap, the idea is quite simple and can be readily applied in almost any circumstance. Briefly, the procedure involves repeatedly resampling the original data after removing any real structure (i.e., randomizing the data) to generate an empirical distribution of the test statistic under the null hypothesis of no real structure. There is no need to assume an underlying theoretical (i.e., parametric) distribution because the distribution is generated empirically through resampling the original data. The basic approach is very simple. All we do is randomly shuffle some or all of the data, taking care not to produce observations that fall outside the domain of the original data. The intent is to remove real structure from the data, and this is usually accomplished by shuffling just one of the variables, although this can vary depending on the context of the test. After generating a random permutation of the data, we calculate the test statistic of interest. We repeat the process over and over, usually 1,000-10,000 times, and the permutation distribution of the statistic provides an empirical distribution of the test statistic under the null hypothesis of no real structure. We compare the original test statistic to the permutation distribution to compute an exact *p*-value.

Note the difference between the randomization test procedure and the bootstrap. With the bootstrap, the original data structure is maintained; that is, the individual observation vectors are left intact. We simply draw intact observation vectors randomly (with replacement) to represent the randomness in the sampling process. With the Monte Carlo randomization procedure, the original

data structure is destroyed by randomly shuffling some of the data. In this manner, the bootstrap generates a distribution of the test statistic under the *alternative hypothesis* (of real structure), which is used to construct the confidence interval for the statistic, whereas the Monte Carlo randomization procedure generates a distribution of the test statistic under the *null hypothesis* (of no real structure) which can be used directly to compute a *p*-value.

The Monte Carlo randomization procedure has many other applications in statistics, but here we will focus on its use in testing the null hypothesis of no real structure. We will illustrate the use of the Monte Carlo randomization procedure in the context of a simple linear regression first and then illustrate a more complex use of the procedure.

3.1 A simple example – linear regression

Let's begin by reading in some data. The example data represent standardized breeding bird counts (bird) and a variety of habitat variables (hab) for 30 landscapes (N=30) in the Oregon Coast Range. See the corresponding metadata file (birdhab.meta.pdf) for a description of the data.

```
birds<-read.csv('bird.sub.csv',header=TRUE)
hab<-read.csv('hab.sub.csv',header=TRUE)
```

For convenience, let's merge the bird and habitat data into a single file by typing:

```
birdhab<-merge(birds,hab,by=c('basin','sub'))
```

In this case, the relational fields that link the bird and habitat data sets are 'basin' (3 unique basins) and 'sub' (10 unique subbasins in each basin). The resulting data frame contains a lot of variables, but for our purposes we are only interested in two: (1) Simpson's diversity index for breeding birds (b.sidi), and (2) Simpson's diversity index for vegetation cover types defined largely on the basis of seral stage. Note, the latter index represents the diversity in *landscape composition* as defined largely by vegetation seral stage.

Let's begin by plotting the data:

```
plot(birdhab$s.sidi,birdhab$b.sidi)
```

It appears that there is a negative relationship between bird diversity and vegetation diversity; specifically, bird diversity declines as the vegetation diversity increases. This seems somewhat counter-intuitive, since we usually think of animal diversity increasing with the diversity of habitats (proxied here by vegetation seral stages). Could this result have been due to chance? Or is this relationship likely to be real? Let's find out the nonparametric way; i.e., without having to assume any particular error distribution for our statistical model.

First, let's fit a trend line to the data using linear regression based on ordinary least squares and add it to the plot. We can do this in a single line if we are a little clever. We embed the regression model inside the abline() function, which will accept a fitted regression object as input using the reg= argument, as follows:

```
abline(reg=lm(b.sidi~s.sidi,data=birdhab))
```

Let's use the same approach to add a 95% confidence envelope for the expected value to the fitted regression line using the ci.lines() function in the biostats library, as follows:

```
ci.lines(lm(b.sidi~s.sidi,data=birdhab),lty=2)
```

Let's also extract the slope coefficient from the fitted regression using the coef() function, as follows:

```
slope.obs<-coef(lm(b.sidi~s.sidi,data=birdhab))[[2]]
slope.obs
```

Note, the [[2]] is used to extract the value of the slope coefficient from the lm() object, which happens to be stored in the second component of the list object.

We would like to know how likely it is for us to observe a slope coefficient this large and negative if in fact there is no real relationship between bird diversity and vegetation diversity. We can use the *t* statistic and corresponding *p*-value computed by the lm() function, but this assumes that the errors are normally distributed about the mean, which may or may not be the case here. To be on the safe side, we are going to construct our own null hypothesis test using a Monte Carlo randomization procedure.

First, let's simplify our data set by extracting just the two variables we need for this exercise:

```
data<-subset(birdhab,select=c(b.sidi,s.sidi))
```

Now, we can create a single random permutation of the data by shuffling one of the variables to remove any "real" relationship between them, but first let's copy the original data to a new object so as to leave the original data unaltered, as follows:

```
perm<-data
perm$b.sidi<-perm[sample(1:nrow(perm)),1]
```

This probably requires some explanation. The square brackets means that we are used indexing. We used the sample() function in the place of the row index to shuffle the row index, and we selected 1 as the column index so that we shuffled just column 1. We assigned the result back to the original data for the variable named b.sidi. Since b.sidi already exists, it will be replace by the result of the right-hand side of the equation. This is a simple way to shuffle a column of a data frame or matrix.

Next, let's plot the data as before, only this time it will be the permuted data:

```
plot(perm$s.sidi,perm$b.sidi)
abline(reg=lm(b.sidi~s.sidi,data=perm))
ci.lines(lm(b.sidi~s.sidi,data=perm),lty=2)
coef(lm(b.sidi~s.sidi,data=perm))[[2]]
```

This is just one permutation of the data set without any structure. We need to do this many times to see what the range of outcomes could be when there is truly no real structure, since any one permutation could by chance have a slope that is different from nearly zero. So, we put this into a loop. First, we will copy our original data set to a new object as before and create objects to hold the number of permutations and the permutation results:

```
perm<-data
m<-10000
result<-numeric(m)
```

Next, we will implement a loop that will create a permuted data set as before and then extract the slope coefficient from the linear regression and save it to the object we created above:

```
for(i in 1:m){
    perm$b.sidi<-perm[sample(1:nrow(perm)),1]
    result[i]<-coef(lm(b.sidi~s.sidi,data=perm))[[2]]
}
```

Let's plot the permutation distribution of slope values and add a vertical line to show where our original estimate of the slope based on the real data falls in relation:

```
hist(result)
abline(v=slope.obs,col='red',lty=2)
```

We can use the quantile() function to determine the slope value for which less than 5% of the permuted values lie, which serves as a critical value for a lower one-sided significant test at the alpha=.05 level:

```
quantile(result,c(.05))
```

If we want an exact *p*-value for this lower one-side test, we can compute the percentage of the permuted distribution less than or equal to our observed slope value:

```
sum(result<=slope.obs)/m
```

There you have it. Convincing evidence that the slope we observed is real; i.e., that bird species diversity declines with increasing vegetation landscape diversity in this system. Why this is so, is another question you can ponder.

3.2 A more complex example -- mantel test

Now that you understand the basics behind using the Monte Carlo randomization procedure, let's try using the randomization test in a slightly more complex application involving testing the correlation between two distance matrices – known as the *Mantel test*. The principle behind the Mantel test is quite simple. Take a set of *n* observations measured on two (or more) sets of variables, in which one set of *p* variables (y-set) is presumed to be the dependent on the other set of *m*

variables (x-set). Thus, we have two matrices with the same number of rows: n x p (y-set) and n x m (x-set). We can transform the two data sets into n x n distance matrices, where the distance matrices record the "distance" between each pair of observations based on the corresponding set of variables. Then we can use a simple measure of association such as Pearson's product-moment correlation or Spearman's rank correlation to evaluate the strength of the relationship between the two distance matrices.

The strength of the Mantel procedure is in the flexibility in defining "distance". Instead of just measuring the Euclidean distance between observations, which is the basis for most conventional tests statistics, we can measure the ecological distance between observations using any of a wide range of distance metrics that have been developed for ecological applications. In addition, we can compute the distance between observations based on one or many variables, so the procedure lends itself well to both univariate and multivariate situations. However, because the elements of distance matrices are not independent, because each observation is represented n-1 times in the distance matrix, we cannot use classical test procedures. Instead, we need to test the significance of the correlation between matrices using a permutation test.

Let's begin with our bird data set from the Oregon Coast range. We have 30 observations (subbasins) with a standardized abundance estimate for 98 different bird species. One question we might ask is how different are the bird communities between any two subbasins? If the two subbasins contain the same set of species with the same relative abundances, they are perfectly the same in community structure and should have a "distance" of 0. On the other hand, if they share no species in common, they are perfectly dissimilar and should have a "distance" of 1. Most will have some but not all species in common and will have differences in the relative abundances of those that are in common, so they will have intermediate distances between 0 and 1. There are many different metrics for measuring distance in ecological communities and we will not review them here. Instead, we will adopt perhaps the most commonly used metric known as *Bray-Curtis distance*, which in simple terms measures the percent shared community structure between two sample observations.

There are several functions in R for creating a distance matrix. We will use the data.dist() function in the biostats library, which is simply a wrapper for the vegdist() function in the vegan library. Thus, we have to first load the vegan library before calling the data.dist() function, as follows:

```
library(vegan)
dist.birds<-data.dist(birds[,4:101],method='bray')
```

Note, because the bird data set contains more than just species counts, we have to select just the variables we want. Also, because there are several distance metrics available in the function, we need to specify the method we want to use, which in this case is method='bray' for the Bray-Curtis distance.

Now we have a 30x30 distance matrix in which the entries represent the Bray-Curtis distances between each pair of subbasins. Take a look at the dist.birds object if you like. Note, that the distance matrix is a square, symmetric matrix; that is, the diagonals are zero (since each subbasin is perfectly similar to itself) and the upper triangle is the mirror image of the lower triangle, because

the distance between subbasin A and B is the same as the distance between B and A.

While there are many interesting questions we might ask about the relationships expressed in the bird distance matrix, one question that often arises is whether community dissimilarity is a function of the geographic distance between samples. In other words, are two communities more alike if they are closer together. This is another way of asking whether there is any spatial autocorrelation in the bird community structure. The Mantel test provides an easy way to examine this question. All we need to do is create a distance matrix based on the geographic distances between subbasins, which we can do as follows:

```
dist.space<-data.dist(hab[,3:4],method='euclidean')
```

Note, here we used the x,y or lat-long coordinates (in columns 3 and 4 of the hab data set) of the centroids of the subbasins and method='euclidean' to calculate the Euclidean distance between subbasins.

Next, we simply correlate the two distance matrices using the now familiar cor() function, with the Pearson product-moment correlation as the default, as follows:

```
cor.obs<-cor(dist.space,dist.birds)
cor.obs
```

What does the correlation say about the level of spatial autocorrelation in bird community structure? Is the correlation large enough to be considered real? It is hard to say without testing the significance of the relationship.

Now, it is tempting to use the cor.test() function as we have before to test the significance of the correlation. Let's do so for kicks:

```
cor.test(dist.space,dist.birds)
```

The result indicates that the correlation is highly significant. However, recall that this test assumes that the observations are independent, among other things, but we know that they are decidedly not. So we must devise another way to test the significance of the correlation. Enter the Monte Carlo randomization test procedure, which has no onerous assumptions.

Let's begin by assigning the number of samples and the number of permutations to objects for convenience:

```
n<-attributes(dist.birds)$Size
m<-10000
```

Note, we could have just as easily assigned *n* based on our knowledge that there are 30 subbasins (n<-30) or we could have assigned it based on the number of rows in the bird data set (n<-nrow(bird)). The approach we used above illustrates how you would extract the size from the distance matrix object.

Next, let's create a vector to store the permutation results. In this case, a one-dimensional vector is sufficient since we are only going to store a single number (correlation) from each permutation.

```
result<-numeric(m)
```

Next, we will implement a loop that will create a permuted distance matrix for one of the two distance matrices (it doesn't matter which) and then compute the Mantel correlation between matrices and save it to the object we created above:

```
for(i in 1:m){
    rand<-sample(1:n)
    temp<-as.dist(as.matrix(dist.space)[rand,rand])
    result[i]<-cor(temp,dist.birds)
}
```

Note, the trick we had to use in the loop above is that we first created a random list of numbers from 1 to n, but without replacement so that each number is used once and only once. This creates a shuffled list of numbers between 1 and n. Then, we reorganized the geographic distance matrix (dist.space) by randomly selecting elements and reconstructing a new distance matrix. The syntax says convert the dist.space matrix to an regular matrix (which contains both the upper and lower triangles instead of just the lower) and then order the entries by row and column index given in the rand object, and then covert the matrix back into a distance matrix.

Let's plot the permutation distribution of correlation coefficients and add a vertical line to show where our original estimate of the correlation based on the real data falls in relation:

```
hist(result)
abline(v=cor.obs,col='red',lty=2)
```

We can use the quantile() function to determine the correlation value for which 95% of the permuted values lie below, which serves as a critical value for an upper one-sided significant test at the *alpha*=.05 level:

```
quantile(result,c(.95))
```

If we want an exact *p*-value for this upper one-sided test, we can compute the percentage of the permuted distribution greater than or equal to our observed correlation value:

```
sum(result>=cor.obs)/m
```

Now that we have a valid significance test, what does it say about the level of spatial autocorrelation in bird community structure?

If you are troubled by the notion of having to write a script like this every time you might want to conduct a Mantel test, you will find it a relief to know that functions already exist to do this for you and in more complex ways, for example when you want to partial out one (or more) distance

matrices before computing the correlation. The vegan library, for example, includes the mantel() function, which does exactly what we just did in one simple call, as follows:

```
mantel(dist.space,dist.birds)
```

So, why did I make you go through the exercise of writing your own Mantel test? Obviously, to show you just how the randomization procedure actually works, because there will be times when existing functions to do what you want don't exist.

4. Exercise

The purpose of the following exercise is to give you additional experience working with the bootstrap and randomization tests. You can work individually or in teams to complete the exercise.

The first problem is to compute *rarefaction curves* for two (or more) different subbasins in the Oregon birds data set. The data represent standardized breeding bird counts (bird) for the 1046 sample plots (N=1046). See the corresponding metadata file (birdhab.meta.pdf) for a description of the data.

1. Read in the raw data, list the unique subbasins, and select a subbasin (which is actually a combination of basin and subbasin since a few of the subbasin names are the same in two basins) of your choice, making sure to select just the bird species variables (columns), as follows:

   ```
   birds<-read.csv('bird.sta.csv',header=TRUE)
   unique(birds$basin,birds$sub)
   data<-birds[birds$basin=='D' & birds$sub=='AL', 4:81]
   ```

   Note, replace 'AL' with the subbasin of your choice.

2. Compute a rarefaction curve for at least two different subbasins and discuss the difference between them in your report. See if you can figure out how to plot the curves together on the same plot, which will make for a better comparison? If you have lots and lots of patience (like overnight), you can try running the script provided to plot the mean rarefaction curve for all 30 subbasins.

The second problem is to construct a *randomization test* for the effect of understory treatment on tree seedling density. The data come from a field experiment designed to assess tree seedling response to three different understory vegetation treatments. We will not concern ourselves here with the nature of the treatments. The treatments (including a control) were randomly assigned to 32 plots in a randomized block design. The blocking factor had no effect on the results, so we will ignore it in this example.

1. First, read in the raw data (vegdata.csv) and plot the number of pine seedlings by treatment to visually see what we are working with.

2. Conduct a one-way analysis of variance with pine seedlings (pine) as the dependent variable and treatment (treatment) as the independent variable. For this exercise you can use the lm()

function wrapped in a summary() function to see the analysis of variance table results. Note the *F*-statistic and *p*-value for the overall test of treatment effect.

3. Next, construct a randomization procedure to test the significance of the treatment effect to avoid the underlying parametric assumptions of the *F*-test. Specifically, construct a permutation distribution for the *F*-ratio statistic under the null hypothesis of no treatment effect and determine how likely you would have observed the original *F*-statistic if the null hypothesis were true. Report the permutation distribution figure and the exact *p*-value for your randomization test. Hint, use the linear regression example as a template. The key trick you will need to know is that the *F*-statistic can be extracted from the summary of the linear model object as follows:

   ```
   summary(lm(pine~treatment,data=perm))$fstatistic[1]
   ```