Introduction to Ecological Analysis in **R** (Day 2)

Matthew Lau

November 4, 2008

1. Integrative Analysis
2. Making Better Barplots
3. Community Data
4. Species Accumulation Curves
5. Visualizing Data

1 Integrative Analyses

When working in R it's helpful to make sure that you manage objects, functions (i.e. commands) and packages effectively. You can do this by:

1. Setting the working directory to a specific folder where you can keep all of the files pertinent to your analysis
    - You can now enter the file name instead of the whole file path when loading data into **R**
2. Promptly removing objects using the `rm` command
    - `rm(list=ls())` removes all objects visible to **R**
3. Detaching packages when you're through with them
    - `detach(package:`*`package name`*`)`

2 Making a Better Bar Plot

The base function `barplot` does not have a argument specification for making error bars, which is the standard presentation for mean data. Thankfully, someone ran into this issue and made a function that does. This function is a part of a package (i.e. a collection of functions) called *gplots* that is available on the CRAN site. To use this function, first we need to load that package from the CRAN site:

```
> install.packages("gplots")
```

Next, we load the package into R:

```
> library(gplots)
```

Now, we can load our data, make our mean and standard error vectors, which we will be using to make our plot:

NOTE: I have set the working directory to a folder containing the data. Thus, I have only given the file name in the `read.csv` **function.**

```
> anova.data <- read.csv("PS_Day1_ANOVA.csv")
> attach(anova.data)
```

```
> anova.means <- c(mean(Bat_Abundance[Fire_Intensity == 1]), mean(Bat_Abundance[Fire_Inten
+     2]), mean(Bat_Abundance[Fire_Intensity == 3]))
> anova.means
```

```
[1]  9.302963 26.189348 15.995588
```

Now we need to create our a vector for our error bars. We do this by first making a vector of standard deviations, which we divide by the square root of our sample size (obtained by the length of one of our response vectors):

```
> anova.sd <- c(sd(Bat_Abundance[Fire_Intensity == 1]), sd(Bat_Abundance[Fire_Intensity ==
+     2]), sd(Bat_Abundance[Fire_Intensity == 3]))
> n <- length(Bat_Abundance[Fire_Intensity == 1])
> anova.se <- anova.sd/sqrt(n)
> anova.se
```

```
[1] 1.221602 2.205972 1.600004
```

We now add and subtract the standard error vector from our means to get our error bar vectors:

```
> anova.ci.u <- anova.means + anova.se
> anova.ci.l <- anova.means - anova.se
> anova.ci.u
```

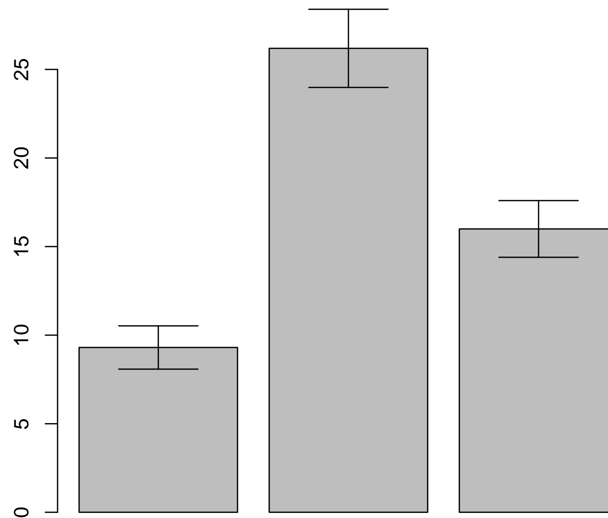
```
[1] 10.52456 28.39532 17.59559
```

```
> anova.ci.l
```

```
[1]  8.081361 23.983377 14.395584
```

We can now make our plot:

```
> barplot2(anova.means, plot.ci = TRUE, ci.u = anova.ci.u, ci.l = anova.ci.l)
```

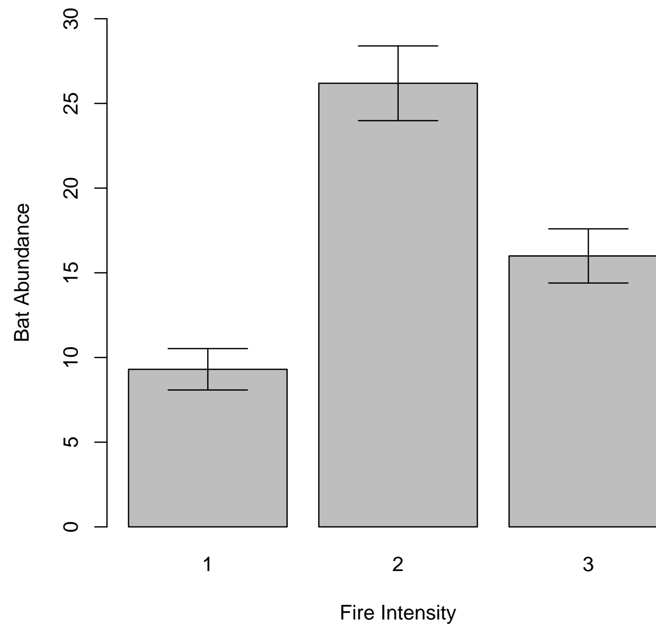


Remember, there are many more arguments in `barplot2`, and just about every other plotting function, that will allow you to specify almost anything you want. For instance, we can improve the plot substantially by just modifying a few argument specifications:

```
> barplot2(anova.means, plot.ci = TRUE, ci.u = anova.ci.u, ci.l = anova.ci.l,
+     xlab = "Fire Intensity", names = c("1", "2", "3"), ylab = "Bat Abundance",
+     col = 8, ylim = c(0, 30))
```

If you look carefully at the argument specification, you can see that to get our error bars we had to specify the logical argument *plot.ci* as TRUE and the *ci.u* and *ci.l* arguments, which define the "upper" and "lower" limits of the error bars, using our error bar vectors. We then also had to specify:

1. *xlab* which is the "x-axis" label
2. *names* which are the levels of x
3. *ylab* which is the "y-axis" label
4. *col* to make the bars grey
5. *ylim* which gives the "y-axis" limits



And now, we clean up:

```
> rm(list = ls())
> detach(anova.data)
```

3 Community Data

Working with community data presents its own set of questions and issues. Here we detail how to:

1. Manage community data
2. Check how well our sample represents the community
3. Obtain univariate community statistics
4. Visualize community data
5. Test for patterns in our community data

```
> com.data <- read.csv("CommData.csv")
```

This file contains both the grouping information as well as the species abundances for each observation in rows. First, we pull out our environmental data and our community matrix:

```
> env <- factor(com.data$env)
> com <- com.data[, 2:ncol(com.data)]
> com <- as.matrix(com)
```

The *as.matrix* function converts our current "matrix" into the matrix format recognized by R (the `read.csv` function imports it as some other format). The dollar sign "$" refers to a particular column (in this case *env*) within a data frame (in this case *com.data*).

4 Species Accumulation Curves

Our inferences from our community analyses rely on whether or not we have adequately sampled the community. To assess this we can create species accumulation curves using functions available in the *vegan* package.

First, we will need to install and load the *vegan* package:

```
> install.packages("vegan")
```

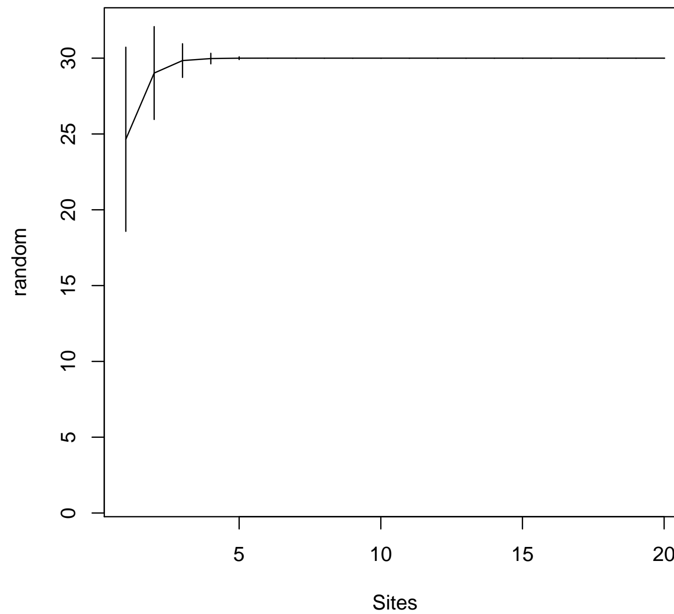
```
The downloaded packages are in
        /var/folders/+U/+UN68GjHFRKn4hmJK1Vh0k+++TI/-Tmp-//Rtmp3Gl9wj/downloaded_packages
```

```
> library(vegan)
```

Now, we will use the `specaccum` function to generate the species accumulation curve information:

```
> spp.curve <- specaccum(comm = com, method = "random", permutations = 1000)
> plot(spp.curve)
```

Notice how we have specified our community matrix, method and permutation arguments. Simply, this tells the `specaccum` function what the data matrix is called, what method to use to generate the curves and how many permutations to use to generate our confidence inter-



vals.

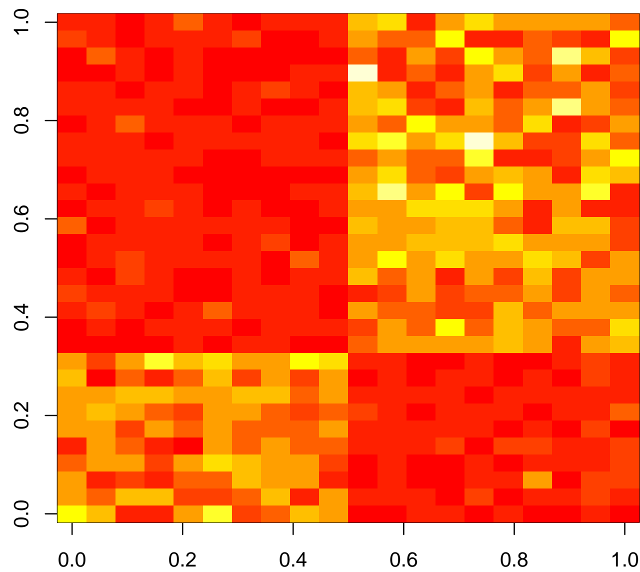
Considering the the the curve levels off way before the number of observations we have, we can confidently say that we have adequately sampled the community. Now we can proceed with our analyses.

5 Visualizing Community Data

5.1 Heatmaps

Heatmaps are typically used to present molecular data (e.g. micro-array data), but I have found them to be a useful look at the entirety of the dataset before proceeding to ordination based techniques. To do this, we simply use the *image*:

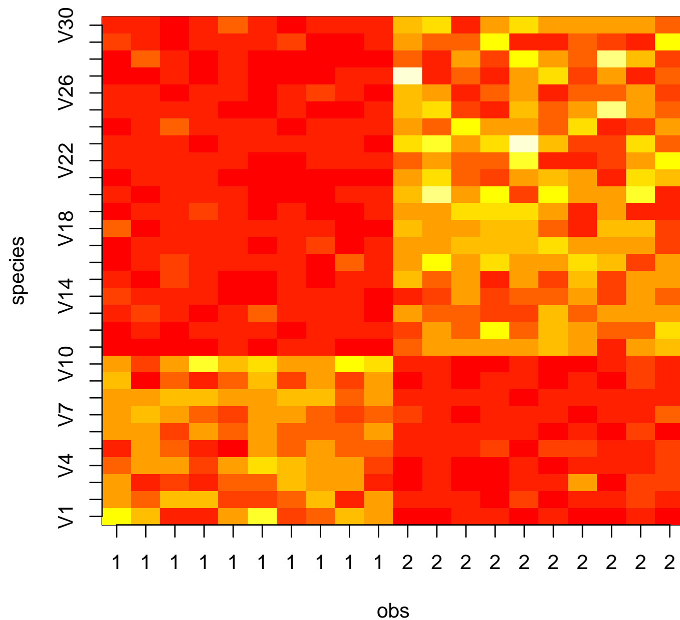
```
> com <- as.matrix(com)
> image(com)
```



You'll notice that we first used the *as.matrix* function on our community matrix. This is because when we imported the data using the *read.csv* function, R formatted it as a "data frame". Unfortunately, some functions, such as the *image* function, require the data to be a "matrix" format. You can check to see if an object is a matrix by using the *is.matrix* function.

Next, we may want to alter this plot in order to make it more comprehensible. For instance, we can change the axis by:

```
> obs = seq(1, nrow(com), by = 1)
> species = seq(1, ncol(com), by = 1)
> image(x = obs, y = species, z = com, xaxt = "n", yaxt = "n")
> axis(side = 1, tick = TRUE, at = obs, labels = env)
> axis(2, tick = TRUE, at = species, labels = colnames(com))
```



We now have a plot with the appropriate lables. What we have done to do this is first make two vectors that define the axis markings for our observations and our species. We then created a plot with no axis labels by specifying the arguments, *xaxt* and *yaxt*, as ”n”, which is short for ”none”. We then use the `axis` **function to plot both the x (observations) and the y (species) axes. For more information on the argument specifications for these functions, please consult the help files (e.g. help(axis)).**

We can also make a legend by adding the script at the bottom specifying the *legend* function:

```
> obs = seq(1, nrow(com), by = 1)
> species = seq(1, ncol(com), by = 1)
> image(x = obs, y = species, z = com, xaxt = "n", yaxt = "n")
> axis(side = 1, tick = TRUE, at = obs, labels = env)
> axis(2, tick = TRUE, at = species, labels = colnames(com))
> legend("topleft", legend = seq(min(com), max(com), by = 50),
+     col = heat.colors(9), pch = c(0, 0, 0, 0, 0, 0, 0, 0, 0,
+         0), bg = "white")
```

5.2 Ordination

The heatmaps are only useful to a certain extent for visualizing community data. The more commonly used method is ordination. This is merely a process of simplifying the multi-dimensional data into

something with fewer dimensions that is presentable and still confers important patterns in the data. Non-metric Multidimensional Scaling (NMS) using Bray-Curtis distance is the most commonly used ordination technique used for community data. This method is robust to typical characteristics of community data. It also represents the data in distance space, preserving the rank order of the observations.

To conduct NMS ordination in R, we will first need to load several packages that contain functions that we will be using:

```
> install.packages("ecodist")
```

```
The downloaded packages are in
        /var/folders/+U/+UN68GjHFRKn4hmJK1Vh0k+++TI/-Tmp-//Rtmp3Gl9wj/downloaded_packages
```

```
> install.packages("BiodiversityR")
```

```
The downloaded packages are in
        /var/folders/+U/+UN68GjHFRKn4hmJK1Vh0k+++TI/-Tmp-//Rtmp3Gl9wj/downloaded_packages
```

```
> install.packages("ellipse")
```

```
The downloaded packages are in
        /var/folders/+U/+UN68GjHFRKn4hmJK1Vh0k+++TI/-Tmp-//Rtmp3Gl9wj/downloaded_packages
```

As before with the *gplots* package, we now need to load these packages into R:

```
> library(ecodist)
> library(BiodiversityR)
> library(ellipse)
> library(vegan)
```

Notice here that I have also loaded the *vegan* package, even though I didn't install it in the last step. This is because once you have installed a package, as we did when we were generating species area curves above, you do not need to install it again (unless you delete it). You do, however, need to load it again if you have detached it or closed and re-startred R.

```
> dis <- distance(com, method = "bray-curtis")
```

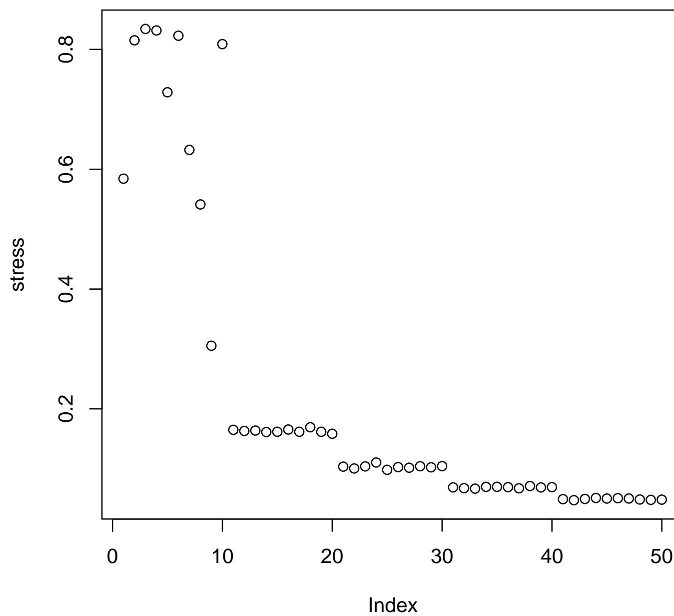
The next step in NMS ordination is to determine the most parsimonious number of axes. That is, we need to find out the lowest number of NMS axes that will adequately represent the data. Here, we can use stress (i.e. the distance of the NMS representation from the original, un-scaled, distances) to assess the departure of the NMS configuration from the actual data.

To determine the parsimonious axis number, we generate a scree plot (for more info, consult the PC-ORD manual). We can do this using the *nmds* command in the *ecodist* package:

```
> scree <- nmds(dis, mindim = 1, maxdim = 5, nits = 10)
```

```
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
```

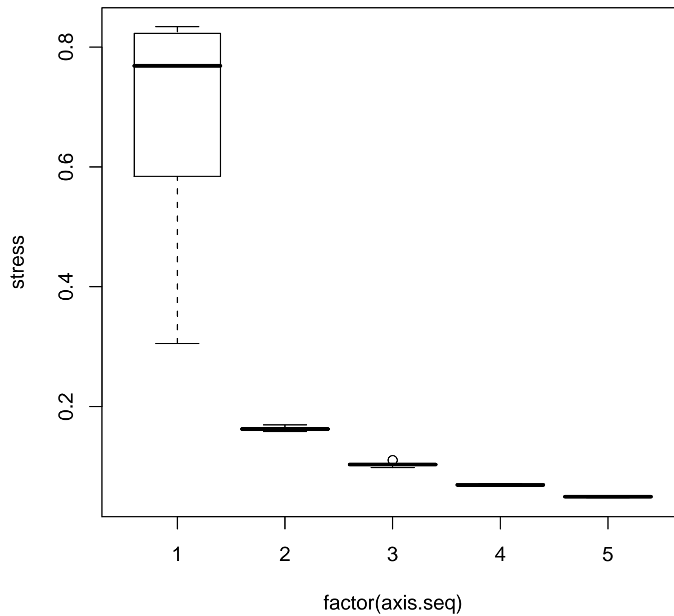
```
Using random start configuration
Using random start configuration
Using random start configuration

> stress <- scree$stress
> plot(stress)
```



This plot is of the stress values from the minimized stress configurations from ten random starts (*inits*=10) at each level of dimensionality, one to five, (*mindim=1,maxdim=5*), ranked in order from the first to the last configuration, which means that the first ten stress values are all from 1-D configurations, the second ten are from 2-D, and so on. What we can see is that the stress rapidly decreases as we increase the number of nms axes and that at two axes we have an acceptable stress level with little variation amongst repeated random starting configurations. Thus, 2-D is fine for our purposes.

Here is the code for a more presentable scree plot (dissect it at your leisure):

```
> axis.seq <- c(seq(1, 1, length = 10), seq(2, 2, length = 10),
+     seq(3, 3, length = 10), seq(4, 4, length = 10), seq(5, 5,
+         length = 10))
> plot(stress ~ factor(axis.seq))
```



We can now re-run the *nmds* command with more iterations to "explore" more of the ordination configuration space and ensure that we have the lowest stress configuration possible:

```
> nms.final <- nmds(dis, mindim = 2, maxdim = 2, nits = 50)
```

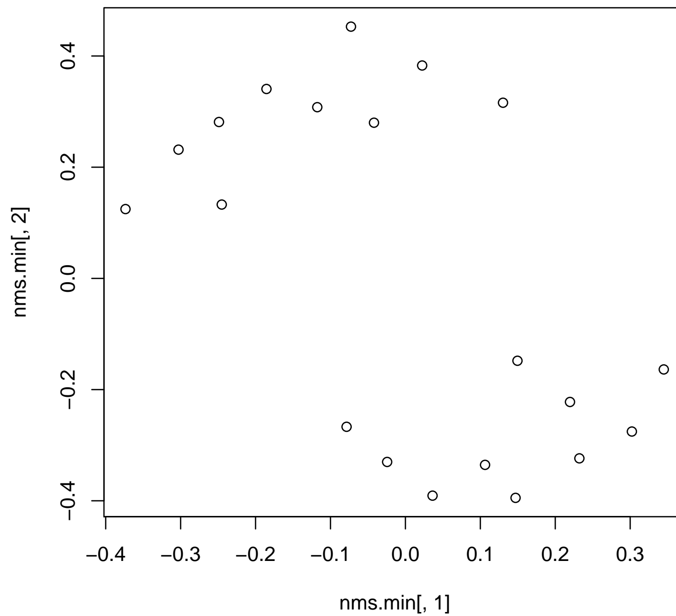
```
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
```

```
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
Using random start configuration
```

Now we have an object containing a whole bunch of minimized stress configurations generated from random starting configurations. We can use the *nmds.min* function to extract the lowest stress configuration and plot it:

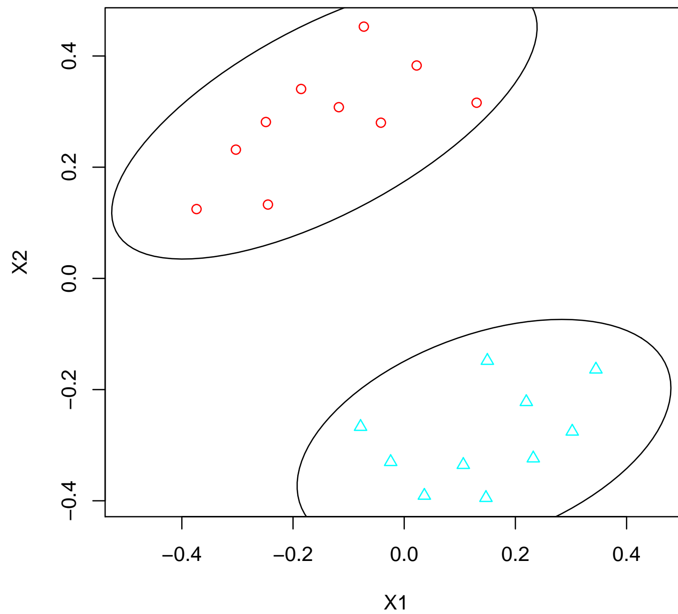
Now we have an object that contains a whole bunch of minimized stress NMS configurations generated from random starts. We can use the *nmds.min* function to extract the lowest stress configuration and plot it:

```
> nms.min <- nmds.min(nms.final, dims = 2)
> plot(nms.min[, 1], nms.min[, 2])
```



This plot is hardly satisfactory. To make a better plot we can use more functions from our packages that are specific to making ordination plots:

```
> nms.plot <- ordiplot(nms.min, type = "n")
> env.frame <- data.frame(env)
> ordisymbol(nms.plot, y = env.frame, factor = "env", rainbow = T,
+     col = env, legend = F)
> ordiellipse(nms.plot, env, kind = "sd", conf = 0.95)
```



Here, we make a plot with no points in it with the `ordiplot` function, saving the plot information into an object for further use in the `ordisymbol` function. Notice the use of the `data.frame` function. This creates a data frame from the object "env", which is the expected format for the `ordisymbol` function. Last, we use the `ordiellipse` function from the *ellipse* package to draw confidence ellipses on our plot.

Last, to get the actual stress value for our final configuration:

```
> min(nms.final$stress)

[1] 0.1565177
```

Next week we will finish up community analysis by going over:

1. Multivariate Analyses (i.e. MRPP, ANOSIM and PerMANOVA)
2. Indicator Species Analysis