Quantitative Ecology, Analytical Lab 2: Similarity and Distances

**Fall 2021**

\*\*\*Look at the assignment at the end first. If you don’t, you’ll wish you had when you get there.

*I. Introduction*

A. Overview

* Earlier we examined Brown and Swan 2010 from Journal of Animal Ecology. This week we’re going to use the data from that study to investigate similarities a little more
* The lab will have 2 basic parts. The first will be just simply fooling around a bit with the similarity functions, learning how to use them. The second will be running some more complex scripts and then using those to interpret the results similar to what we did in Brown and Swan.
* A couple of secondary goals today involve you guys becoming a bit more familiar with R, particularly with scripts and how to use them for more complex analyses. You won’t need to write your own, but you will need to understand them enough to make some simple modifications

B. Materials

1. Datasets

\* All data come from the Maryland Biological Stream Survey (MBSS) dataset. It is a public dataset

collected by the state of Maryland used in Brown and Swan 2010.

* **‘YSC Inverts.csv’** = This is one of two major datasets that you’ll be using today. The “YSC” stands for Youghiogheny, Savage, and Casselman, the 3 drainages included in the Brown and Swan analysis. If you look at the data, you’ll find that the matrix consists of 7 columns of identifying information (e.g., Site, Drainage, Stream Order), and then the rest of the columns are benthic macroinvertebrate taxa.

It’s also worth mentioning that these data have been reformatted from their original form. So if

you want to go grab and use the MBSS dataset yourself, you won’t get a pretty matrix like the

one we have today. If you find yourself in that position and don’t feel like reinventing the

wheel, come and talk to me.

* **‘Distance.csv’** = File containing the distances between pairs of sites. The file includes 2 types of distances, Euclidean distance and stream network distance. The file also includes the difference between those two types of distances, but that won’t come into play in today’s lab.
* **‘Codes.csv’** = Basically just a file giving identification information allowing us to link the specific sites to distances between sites. For our purposes, this file will work in the background, though if you’re curious, you might investigate the file and how the scripts use the file a bit.

2. Scripts

* **‘Lab Script-A.R’** = A very basic script to get you started today. Just contains a few lines to read in the YSC Inverts.csv file and then gets you started calculating similarities
* **‘Lab Script-B.R’** = A more complicated script. The script takes the 3 data files mentioned above, uses them to calculate similarities between sites, then merges those similarities with the distances from the Distance.csv file. We’ll step through this script in the lab a little later so that you can try to understand what each part is doing.
* **‘Lab Script-C.R’** = Another fairly simple script that loads the results produced by “Lab Script-B.R”. Once those are reloaded, you can much more easily see the results. It produces a simple figure as well.

*II. Simple Similarity Functions*

A. Examining the script

* Open the script “Lab Script-A.R”
* The script is simple, with only a few coding lines. At this point, you should be familiar with the setwd() function and the next line which imports the dataset.
* Also note the library(vegan) line that loads the vegan package which is where the similarity functions for today live.
* The next two lines are actually performing basic similarity comparisons using the function vegdist(). As indicated by the script annotation, one of those lines compares similarity between two observations in the dataset, while the other produces a similarity matrix based on the entire dataset.
* Finally, we end with a line that clears your R workspace. Run them when you’re finished with this part of the exercise in order to keep your workspace uncluttered. Clutter can become a problem during more complex analyses, especially if you’re not creative naming variables (like me) because you can accidentally overwrite previously-created variables.

\*\*A word on other similarity functions. One thing you’ll quickly become familiar with in R is that there are multiple ways to skin almost every cat. People write new packages frequently and often they include functionality that is largely redundant with other packages. For example, vegdist() is not the only function that performs distance calculations. The function dist() is in the base package and does much the same thing. We’re choosing vegdist() because the functions included in vegdist() are more focused on ecology, but realize that for many operations, there are a lot of different functions out there.

B. Performing pairwise similarity

* Submit the first several lines of the script, up through the library(vegan) line. Those should load the dataset, set the working directory, and load vegan. If you haven’t yet downloaded the vegan package, you may get an error. If so, you’ll need to install the package.
* Using the R-help, examine the vegdist() function. ?vegdist
* The vegdist function has several options or arguments. **Describe each argument/option and what it means.**

vegdist(x, method="bray", binary=FALSE, diag=FALSE, upper=FALSE,

na.rm = FALSE, ...)

x: dataset imported into R

method: dissimilarity index (i.e. difference between community compositions, lower values meaning communities are more similar), chosen by the user from any number of dissimilarity indices (e.g. bray-curtis, jaccard, etc…)

binary: Presence-absence standardization, whether data values should be converted to reflect only whether the species was present or absent.

diag: Computes diagonals. I am not sure what this and upper are actually doing as I was not able to see any change in output just running the vegdist() with these switched on an off. My guess would be that diagonal values (i.e. matrix points where a variable is compared to itself) are removed from the matrix, but again, I was not able to accomplish this.

upper: Returns the upper diagonal only. Again, I am not sure.

na.rm: Typically removes NA values from a dataset. NA values could mean data were missing.

* You may notice that we’re not specifying all of the options. **Which options are we not invoking (you don’t need to make a whole list; just examine them)? Why? What happens to the options you don’t specify?**

Those dealing with diagonals. I assume their default is to be FALSE unless otherwise stated.

* Now, look at the next two lines, the ones that perform similarity analyses. One is pairwise while one creates a matrix. **Why do the two lines function differently?**

One is only calling two variables (2 columns) in the dataset, the other incorporates all variables, excluding those that are not abundance data.

* Now, run the line that creates dist.a that performs a simple pairwise similarity. **Where are the results?** Once you’ve examined the results, **What was the result? What does the result mean?**

Stored within an object ‘dist.a’ that can now be found in the global environment. The result was 0.926, meaning there is little similarity between communities where these species are either both, one, or neither present, and that they are highly variable.

* One critical thing you should always know is whether you’re calculating similarity or dissimilarity/distance. **Which is it here? Does knowing that change your perception of your results?**

Dissimilarity. No, a high value for dissimilarity would mean these communities are dissimilar; likewise, a high value in a similarity matrix would mean communities are similar--two sides of the same coin.

* Now, that you’ve examined how this function works, I want you to play around a bit with it. If you’re already very familiar with this function, then you can move on, but for those of you that aren’t familiar, spend some time with this part. Creating dissimilarity matrices is foundational to the course so being familiar with how these functions work is critical. Some things you might play with a bit:
  + 1. Change which observations from the dataset you’re performing similarity on. Examine the raw data as you do this operation. **Do the results make sense?**

Yes. I chose columns 47 and 33 because they shared the lowest dissimilarity value, then looked at the raw data. These species are typically not present or present together, albeit in similar low numbers of abundance.

* + 1. Change the distance metric. There are several. There are also descriptions in the vegan documentation about what each of them does.
    2. Specify binary=FALSE. See what happens. Keep in mind that you should do this with some metrics and not with others. For example, Jaccard is designed to run on binary data. What happens if you run Jaccard on non-binary data?

C. Creating a similarity matrix

* Now submit the line that creates a similarity matrix. **What did it create? Anything interesting about this object?**

Values are stored as ‘dist’.

* Choose one of the elements in this object and try accessing it directly (i.e., square bracket/matrix notation). **Did it work? What happened?**

Calling a single value failed. Values stored in a ‘dist’ vector cannot be directly called or are not stored in a matrix format.

* As you hopefully discovered, the result of the line that created a similarity matrix may not have been what you were expecting. When R calculates similarities, it creates a *distance object*. That object has attributes beyond what you can see with the naked eye. Why is this desirable? Well, because other functions can “inherit” those properties. For example, when running an NMDS, you start with a distance matrix, so all you need to do is pass the distance object to an NMDS function, and you’re golden (e.g., metaMDS(dist.b); try it if you want. It will work!).
* For our purposes, and often when dealing with real data, you’d rather see a distance matrix as, well, a matrix. Use the as.matrix() function to convert dist.b to an actual matrix. Ahhhh, much better, no? **Can you directly access the elements of this converted matrix?**

Yes, dist.b[4,5] returns the value 0.5897436

* You may also play around with this function a bit if something intrigues you. However, you’ll find that it’s simply the n-dimensional extension of what you were doing before.

*III. Getting more complex*

A. What’s the point?

* Now we’re going to examine the script ‘Lab Script-B.R’ and see what’s going on there
* In Brown and Swan, we were interested in examining the relationship between community similarity and distance between sites. So we had to pull together data of multiple types. What ‘Lab Script-B.R’ does is to perform those operations for you.
* We’re going to step through ‘Lab Script-B.R’ and hopefully most of you will develop a basic understanding of what is going on.

B Examining the script

* The first few lines here should be very familiar to you, setwd(), and the lines that import data. **How many datasets are we importing? Why that many?**

3 datasets: YSC Inverts, Codes, and Distance.

Species abundance data, site information, and network distances are all in different spreadsheets.

* The next section might look a little strange. These 3 lines surrounded by hedges of # are options for this script. Two of those options “Metric” and “Bin” should be familiar to you from vegdist(). A 3rd option is related to how we treat the distances in the analysis.
* Let’s look at how these options work a little more directly. Go down to line 49. Buried in that mass of code is a line that should look familiar, our old friend vegdist(). Examine that line. **How do the options above change how this vegdist() operates?** **Why would specifying options this way be helpful?**

We have already defined our arguments in the environment. I am not sure, as typing sim <- vegdist(ab, method='bray', binary=FALSE) produces the same results. *Having read further, it appears our arguments are defined beforehand to minimize coding errors?*

* OK, now let’s look at the meat of this thing. Drop down to the next few lines below the options. This one is pretty straightforward, just isolating some columns from the dataset. **Why did I do this?**

When the loop is run all together, vegdist() is run on just two lines at a time within a stream order, and then compiled.

* The next part might get a little hairy. I’m happy to walk through this with anyone interested, but I’m going to give you a basic summary of what is going on here. This part may look complicated, but at its heart (line 49) is the simple vegdist() function, and if you pick apart the script, you’ll find that I am using the pairwise version. All of the rest of that business is simply multiple loops that serve two purposes: to isolate the data of interest and then to help combine the similarities it produces with the appropriate distance information. There are 3 loops involved here, and each loop is easy to spot because it begins with the for() loop function:
  1. Subsetting by stream order as in Brown and Swan
  2. Subsetting by Drainage
  3. Running through each pair of sites within a drainage to calculate pairwise similarities between each, then to pair those similarities with appropriate IDs linking them to the distances in the Distance.csv file

This bit of script may look intimidating, but most of you will feel comfortable with this sort of thing before the end of the semester (yes, you will).

* Computationally these nested loops aren’t exactly efficient. A real programmer would probably box my jaws and sit me down in the corner to think about the error of my ways. However, sometimes computational time is less important than writing a straightforward script that is easy to dissect and modify. Thus was my purpose here.
* Step down to the next couple of lines after the loops (the “}” signals the end of the loop). These lines are simply producing the output datasets from that previous set of operations.
* Finally, this last big section of script (starting on line 62) is just using one relatively simple loop to standardize the distances within each watershed as described in Brown and Swan
* And then at the end, we have those nifty little lines to clear the workspace

C. Running the script

* This is a script designed to be run as a whole, not line by line.
* It produces an output dataset that we can play with later, but this script is not designed to be monkeyed with much *with the exception of the options*. It’s one reason for setting the options off loud and clear with our hedges of ###### so that only the options get monkeyed with and nothing else.
* To run an entire script (that is *source* it), you can choose the “source” option in R-Studio’s Code menu. You can also be slick and use Ctrl+Shift+S
* So do it already! It will take a little while to run (about 15 seconds on my office desktop). That’s where the computationally inefficient bit comes in. A real programmer could probably make this run in about 4 seconds, but were those extra 11 seconds really worth the effort? The little stop-sign in the Console window indicates that R is busy. **So what happened?**

Our dataset was merged with standardized mean Euclidean and network distances and saved as an .RData file.

* You might have noticed that nothing happened, or at least nothing that you could see. Look at the last couple of lines of code and answer **what happened?** again.

These last lines cleared our global environment.

*IV. Lets look at the results!*

A. The script

* The script ‘Lab Script-C.R’ is another pretty simple script. This one loads the results from the previous script (Lab Script-B.R) and then draws a simple figure.
* Let’s look at this script quickly.
* Again, the setwd() should be familiar. You should also see, based on how we’re using these scripts, how useful it can be.
* The next line may be a little new. The function should be obvious—to load a dataset—but the method might be unfamiliar. **How does this method of loading data differ from what we’ve seen before? Why did I use this method in this case?** Please take a little time to answer these questions if you don’t already understand the answer. If you can’t find an answer quickly, then ask. The difference is fairly important.

Previously, we were loading a .csv directly and needed to indicate how data should appear in our global environment (as a data frame) and things particular to our data file (having comma separated values, columns having a heading). In the last script, we saved our dataframe as an .RData file, which can now just be pulled into the global environment with the load() command. Now that our dataset has been created, we no longer have to run the previous script to once again combine our three data sets and loop functions through them.

* Examine the data.frame that the previous line loaded. **What does it look like?** HINT: The function head() is great for examining a dataset because it gives you the first 5 lines of a dataset, allowing you to see the structure and variable names. So in this case head(ALL) would do that.

It looks just like the file we created in the ‘B’ script.

* The final few lines draw a figure. Let’s look at these a bit more carefully
  1. The first line is a par() statement. par() sets global graphics parameters. In this case, I’m using one par() option, mfrow(), to tell R to make a 2-panel figure. You can look at help for par() by using ?par to see what other options are there.
  2. The next line is a plot() statement. You’ve hopefully played around with these already. While this statement is simple, it’s also very powerful. You can do a lot with a simple plot() statement. **What data are being plotted here?** Pay attention to the subsetting.

The first plot is just of our first order systems. The mean network distances are plotted against community similarity values.

* 1. This line uses the function lm() to perform a simple linear regression so that we can draw a best-fit line on our figure. Technically, the regression results shouldn’t be reported because they violate one of the major assumptions of linear regression, though we can certainly use them to draw a line. **Why are the actual statistical results not really kosher?** Think about what kind of data these are…

Pairwise data, inflated data numbers

* 1. abline() uses the regression coefficients to draw the line
  2. Basically the same 3 lines repeat to draw a second panel. **But what’s different here?** Look closely, it’s subtle.
* This script is also designed to be run all at once. Since it inherits the results of Lab Script-B.R, changes made there will cascade to here.
* So run it already. **What do you see?**
* One last bit is that you can actually see the results of the regressions. Again, there are some issues here with the actual reliability of these stats (hopefully you figured out what in #3 above), but for our purposes, they’ll be fine for now.
  1. The results of the first regression are saved in ‘reg.1’. If you type ‘reg.1’ in the console, you get some information.
  2. However, if you want the statistical results, you should use *summary(reg.1)*. This behavior in R is pretty standard when looking at stored statistical results.
  3. You can also use *names(reg.1)* to look at all of the information stored in the variable reg.1, and you can access that information using the $ notation. For example reg.1$coefficients will give you the regression coefficients. **What are the regression coefficients for the two graphed relationships?**

Assignment: Due Next week

I want you to actually answer all of the questions in bold in the lab. You don’t have to do it on the actual document if you don’t want. However, these bolded questions are one of the best guides to what’s actually important here, both from a statistical and from a programming perspective. Your answers don’t need to be sophisticated or lengthy, but answer the questions. This format will also be typical of labs in the course, so get used to answering those bold questions, even if they’re not a formal assignment in the future.