7a. Data wrangling with simulated communitites

Z620: Quantitative Biodiversity, Indiana University

OVERVIEW

Much of the power of computing stems from the efficiency of performing small tasks in an iterative manner. In this lesson, we will explore different ways of manipulating or "wrangling" data with the goal of making it easier to perform relevant statistics and generate figures. Specifically, we will introduce traditional **control structures** (e.g., for loops), along with alternative and commonly used tools that are unique to the R statistical environment, such as `apply` functions and `tidyverse` packages. We will learn about these tools while also gaining exposure to R packages that allow us to simulate and sample biodiversity.

1) SETUP

A. Retrieve and set your working directory

```
rm(list=ls())
getwd()
setwd("~/GitHub/QB-2021/1.Handouts/7.ControlStructures")
```

2) SIMULATE BIODIVERSITY

To reinforce principles of alpha diversity, we are going to use the `mobsim` package, which was designed for the simulation and measurement of biodiversity across spatial scales: https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.12986. Simulation is an important and powerful tool, not only in the life sciences, but also a range of other disciplines. It allows one to produce null models, generate hypotheses, and challenge our intuition about how the world works.
In the context of biodiversity, simulations are often initiated by randomly drawing individuals from a *source community*, which is intended to reflect the regional pool of species. In this way, we can build **local communities** while controlling for the number of individuals (N) and species (S) that we will then sample. Let's start by loading `mobsim`, along with other packages. Given part of the focus of this module is control structures, you are going to load the package with a **for loop**. In the code below, the `require()` function in `R` returns `TRUE` if a package was successfully loaded or `FALSE` if the package failed to load. This loop loads each package and installs the package when `require()` returns `FALSE`:

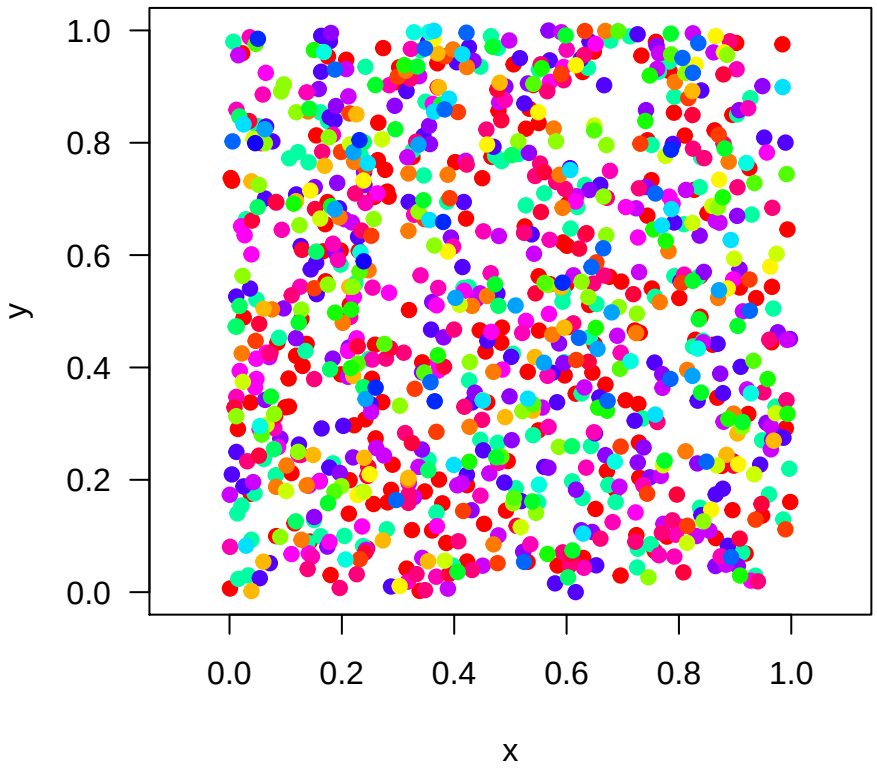
```
package.list <- c('mobsim', 'knitr', 'vegan', 'tidyr', 'dplyr', 'ggplot2', 'formatR')
for (package in package.list) {
  if (!require(package, character.only = TRUE, quietly = TRUE)) {
    install.packages(package)
    library(package, character.only = TRUE)
  }
}

# set page dimensions for printing
```

```
opts_chunk$set(tidy.opts = list(width.cutoff = 70),
        tidy = TRUE, fig.width = 5, fig.height = 5)
```

A. Simulate source community with random spatial positions

Imagine that we want to characterize the diversity of a plant community in an old-field grassland. The site contains 1,000 individual plants (N) belonging to 25 species (S). The individuals in this site will be assembled from a source community that has a log-normal *species abundance distribution (SAD)* with mean = 2 and standard deviation = 1.

```
# simulate community with 'mobsim'
com1 <- sim_poisson_community(s_pool = 25, n_sim = 1000, sad_type = "lnorm",
    sad_coef = list(meanlog = 2, sdlog = 1))
# visualize site
plot(com1)
```

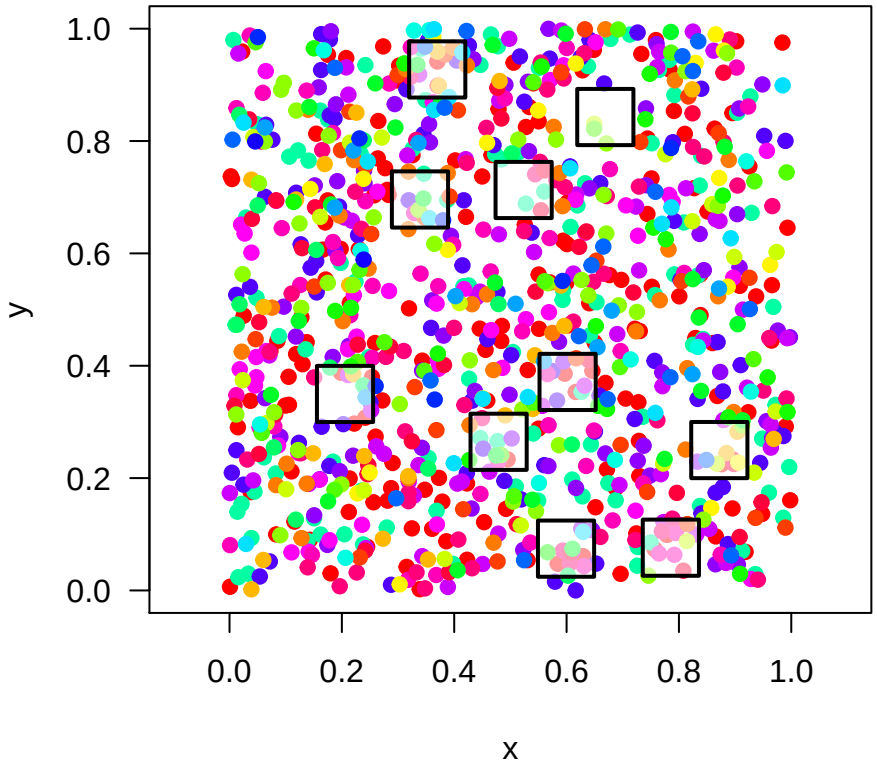


B. Sample the simulated community

In the alpha diversity handout, we emphasized the importance of **sampling effort** and **coverage** when trying to accurately estimate biodiversity. In only a few cases are biologists able to fully census all individuals,

even in relatively simple and tractable systems. As a result, plant ecologists often survey a subset of the individuals within a site using grids, transects, or quadrats. Here, we will used the `sample_quadrats` function to collect individuals from the simulated site. Specifically, we will randomly sample 10 different quadrats, each with area size of 0.01 units.

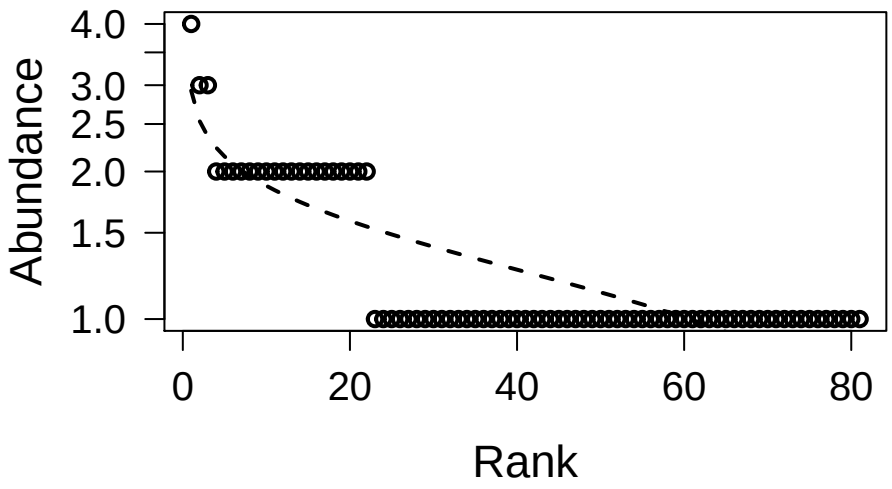
```
# Lay down sampling quadrats on the community
comm_mat1 <- sample_quadrats(com1, n_quadrats = 10, quadrat_area = 0.01,
    method = "random", avoid_overlap = T)
```




```
# Rename sampled areas as quadrats
quads <- c("quad1", "quad2", "quad3", "quad4", "quad5", "quad6", "quad7",
    "quad8", "quad9", "quad10")
row.names(comm_mat1$xy_dat) <- quads
row.names(comm_mat1$spec_dat) <- quads

# Examine the species x quadrat matrix print(comm_mat1$spec_dat)
```

Let's quickly take a look at the rank-abundance curve (RAC) of our plant community using the `rad` function from the `vegan` package. Do the samples fit a log-normal distribution like that of the source community?

```r
# Fit log-normal model to the quadrat-sampled data
RACresults <- rad.lognormal(comm_mat1$spec_dat)

# Print parameters for log-normal fit
print(RACresults)

# Plot log-normal fit to quadrat-sampled data
plot(RACresults, las = 1, cex.lab = 1.4, cex.axis = 1.25, lty = 2, lwd = 2)
```



C. Quantify diversity of simulated community

Now, we will calculate richness, Shannon's diversity, and inverse Simpson's diversity for each of the 10 quadrats. With this, we will create a data frame that summarizes the diversity measures.

```r
# Create an empty data frame
comm_mat1_diversity <- as.data.frame(matrix(ncol = 3, nrow = 10))

# Add column and row names
colnames(comm_mat1_diversity) <- c("richness", "shannon", "invsimpson")
rownames(comm_mat1_diversity) <- rownames(comm_mat1$spec_dat)

# Fill data frame with diversity measures using 'vegan' functions
comm_mat1_diversity[, 1] <- specnumber(comm_mat1$spec_dat)[1:10]
comm_mat1_diversity[, 2] <- diversity(comm_mat1$spec_dat, index = "shannon")[1:10]
comm_mat1_diversity[, 3] <- diversity(comm_mat1$spec_dat, index = "invsimpson")[1:10]

# Examine summary of diversity measures across quadrats
# print(comm_mat1_diversity)
```

2) WRANGLING SIMULATED DATA

A major hurdle in data science and quantitative biology is learning how to efficiently and reproducibly "wrangle" large date sets. This might involve converting your entries from long to wide formats, or subsetting a data matrix based on the values found in a certain column. Or, maybe you want to summarize a data set by applying a function across rows of your matrix. These types of operations are fairly standard when taking classes in programming or computer science, but are less commonly taught in as part of a life-sciences curriculum.

A. Summarize diversity data with a foor loop

In this section, we will introduce some of the basics of **control structures**. Specifically, we will focus on using **for loops**, which have two parts. First, the header assigns a counter or *index* for the variables that will be acted upon. Second, the body contains instructions or operations that will be performed during each iteration. Let's write a for loop to calculate the mean ± standard error (sem) for the abundance of each species in our sampled quadrats:

```
# function for sem
sem <- function(x) {
    return(sd(x)/sqrt(length(x)))
}

# loop to calculate the mean and sem for each row (quadrat), one at a
# time.
for (i in 1:10) {
    x = as.numeric(comm_mat1$spec_dat[i, ])  # each row subset from the species x quadrat
    x = x[x > 0]  # ignore taxa not sampled at each quadrat
    comm_mat1_diversity[i, 4] <- mean(x)  # calculate mean and save
    comm_mat1_diversity[i, 5] <- sem(x)  # calculate sem and save
}

# Rename columns
colnames(comm_mat1_diversity)[4:5] <- c("mean_sp", "se_sp")

# Examine summary of diversity measures across quadrats
# print(comm_mat1_diversity)
```

B. Summarize diversity data with the `apply` function

Although for loops and other control structures (e.g., while loops, if-then) are commonly used by computational biologists, there are some downsides. People argue that they lack clarity and can easily break. This can create confusion when the goal is for code to be robust and reproducible in collaborative projects. Also, in R, control structures can be slow and inefficient. Fortunately, there are some alternatives, that we'll introduce. Within the base R package, there is a family of *`apply` **functions. This allows one to apply functions across row and columns of matrices, lists, vectors, and data frames. The structure of the `apply`**** function differs from the for loop. In fact, it can be done with a single line of code, but it has a similar wort of logic. First, you specify the data object of of interest. Second, you specify the margin that the function will operate (row =1, column = 2, cell = 1:2). Last, you specify the function to use, which can be a native, built-in function (e.g., `mean`) or one that you have written yourself (e.g., `sem`) Run the following code and compare to findings generated with for loop:

```
# if only interested in mean, can just write `mean` at end of function
# for both mean and sem, here, we specify as follows:
comm_mat1_apply <- apply(comm_mat1$spec_dat, 1, FUN = function(x) {
    c(mean = mean(x[x > 0]), sem = sd(x[x > 0])/sqrt(length(x[x > 0])))
})

# transpose so quadrats are rows
comm_mat1_apply <- (t(comm_mat1_apply))
```

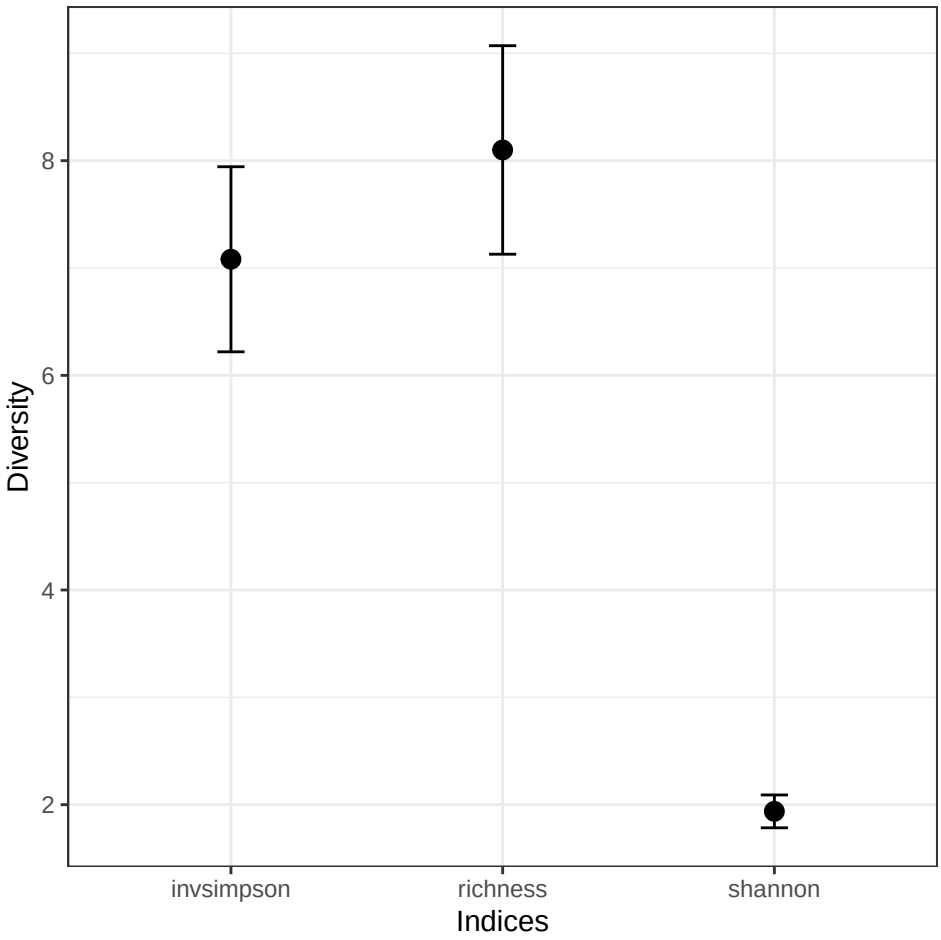
C. Summarize diversity data with `tidyverse`

Another popular approach for data wrangling is **`tidyverse`**, which is a collection R packages. **`dplyr`** contains a set of functions that manipulates data frames. **`tidyr`** allow one to pivot, nest, and separate data in various ways. And **`ggplot2`** is an alternative to plotting in base R that was designed to interface with tidyverse. `tidyverse` uses the **pipe operator**, depicted as %>%, to string together a sequence of functions or operations. Below, we will use the pipe operator in combination to `shape` and `summarize` our diversity data, before making a plot.

```
# wrangle data
comm_mat1_tidy <- comm_mat1_diversity %>% # identify data frame of interest
  select(richness, shannon, invsimpson) %>% # select columns of interest
  pivot_longer(cols = c(richness, shannon, invsimpson),
               names_to = "indices", values_to = "value" ) %>% # convert to long format
  group_by(indices) %>% # group according to these factors
  summarise(mean = mean(value), sem = sem(value)) # calculate mean and sem
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
#print(comm_mat1_tidy)

# plot mean and sem for alpha diversity indices
# create empty plot with information for x and y axes
# by using "+", can layer plot features
ggplot(comm_mat1_tidy, aes(x = indices, y = mean)) +
  # choose plot type (e.g points, lines, boxplots)
  geom_point(size = 3) +
  # add error bars
  geom_errorbar(aes(ymin = mean+sem, ymax = mean-sem), width = 0.1)+
  #relabel axis labels
  labs(x = "Indices", y = "Diversity")+
  # modify plots with "themes"; can be saved to your preferences and re-used
  theme_bw()
```



3) Wrangling a macroecological "law": the species area relationship (SAR)

The species area relationship is one of the most commonly documented patterns of biodiversity. It describes how the number of species recovered changes with increasing area sampled. For nearly all taxonomic groups, species richness (S) increases **non-linearly** with area (A), often fitting a **power law** function: $S = c\ A^z$, where c is a constant and z is describes the rate at which richness scales with area. When richness and area are $\log_{10}$-transformed, the SAR often takes on a linear form. However, the z-values of the SAR vary quite a bit among taxa. For example, in bacterial communities, z-values are relatively low (0.05), while in plant communities they can be much higher (0.25). This variation has led researchers to investigate the factors that influence the SAR. Perhaps it's not surprising that S increases with A. Maybe the pattern is statistically inevitable. But the SAR is also influenced by biological phenomena including dispersal limitation and environmental heterogeneity.

A. Construct a SAR from simulated data

1. Using the `sim_poisson_community` function from the `mobsim`, along with any of the data wrangling tools described above, construct a species area relationship (SAR). Make a plot of the data and calculate the z-value.

2. The `sim_poisson_community`function simulates with species that have a random spatial distribution. However, in nature, this is rarely the case. Species often have patchy distributions, which can be

simulated in `mobsim` using the the `sim_thomas_community` function. Include a plot of the patchy distribution. Then, construct a new SAR with the patchy community and compare the z-values to the random distribution.

3. Explain the SAR patterns and any observed differences based on how we have simulated the communities. Are there any features that you might include with a more elaborate simulation? Describe.