R bootcamp

Julio A. Rivera

R is a statistical programming language that can be used for manipulate data, running stats, making figures, and sourcing other programs to run operations (lime MrBayes for running phylogenetic analyses)

First, let's talk about **scripts**, a script is a bunch of code, usually for a related purpose, in one "document". You will soon write your own fist script in R. When writing an R script, you want to name your script with the extension .R for example: myscript.R

Use simple notepads to write scripts because programs like " Microsoft word" have hidden code that will screw up your code. If you are on a PC, I recommend Notepad ++, if you are on a Mac, I recommend TextWrangler.

To download R, google R CRAN and navigate to the download page. R is open sourced which means it is free to use by anyone. R is built on **packages**, packages are little bits of code that other people have written to make our lives easier. BUT, because is it free and people can just build a piece of code and send it in, you have to always make sure that the code is doing what you think it's doing. This is easier said than done. For now, we will just assume that all the code is OK but when you are working on your own data, always triple check the code.

You really do not use the mouse a lot when coding in R, so liberate yourself from the mouse. Almost all of the work is done in the R console, which is the blank screen that appears when you open R. You will notice a blinking curser, that is where you start typing.

Syntax

Code is read from left to right, like we read a book, therefore, you should code from left to right. Like in math, R is also read from inside the parentheses out (remember PMDAS) so make sure you are organized. Below, we will go over a few important pieces of syntax, but these are in no way a full list. Just something to get us started. We will introduce more syntax as the tutorial goes on.

A few helpful rules when it comes to **syntax**: arrows are really important to syntax **"<-"** or **"->".** Arrows indicate the direction the code is going to be saved, more on this later.

The **equals sign "="** is also important and serves the same purpose as arrows but can be ambiguous and if not coded correctly, bad things could happen. Personally, I only use arrows <- or ->.

Pound signs "#" (or hashtags for you youngsters) mean that anything to right of the # is a comment and will NOT be run as a piece of the code. These are useful for making notes to yourself or collaborators directly in the code.

A **colon ":"** is another important piece of syntax that means through. For example if I write 1:5 this really means 1, 2, 3, 4, 5 or 1-5.

When you have a string of variables (like a list of numbers), they are always separated by a **comma ","**. For example, if I want the number 1 through 3 I can to type 1, 2, 3.

R is also **case sensitive**, so keep track of upper and lower cases.

Function

Functions are pieces of code that have been written by people and simplified into one command. Function names are always followed by parentheses (R doesn't care about spaces with a few exceptions, this is one of them, you should NEVER have a space between the command and the parentheses), that's how we know they are functions. Also, for every open parentheses you need its close parentheses partner. So if you have 3 open parentheses you should have 3 closed parentheses. These should also be in the correct place in the line of code (think PMDAS) or you will get an error. For example, if you type in the following function:

```
mean(1:10)
```

What is returned? The **mean** of 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 which is 5.5

Another important function is the **c** command – this stands for combine. For example

```
c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

What is returned? A list of the numbers 1 through 10.

Functions also have **arguments** and each function will have its own unique set of arguments. These arguments allow you to manipulate the function so it does something more in addition to what you need it to do. Let's say we have the following data 1, 2, 3, and NA. NA might be a piece of missing data you could not get. Now you want to know the sum of this:

```
sum(1, 2, 3, NA)
```

We get an error message because there is a missing value. We can still work with this if we know the correct argument.

```
sum(1, 2, 3, NA, na.rm=TRUE)
```

The **argument na.rm** means that the function removes the NA's before doing what you asked me to. If **TRUE**, the function will remove the NA and do the sum, the default is normally **FALSE** so it doesn't do it automatically.

<u>Objects</u>

What is an object? An **object** is the name given to a variable of some sort. For example:

```
x <- 2
```

This means that x (the new object) is the number 2. This is indicated by the direction of the arrow. This code means that "2" should now be called "x".

But remember that **syntax** is important and you can't define things that already mean something. For example:

```
2 <- x
```

You should get an error because we are trying to redefine the number "2" as "x" which doesn't make sense.

An **object** can also be a **string of variables**, let's try it out by making an object called "y" the numbers 1 through 5 using the c command.

```
y <- c(1, 2, 3, 4, 5)
```

So now the **object** called "y" is the **string of combined** numbers from 1 through 5.

R is made for stats so let's see what we can do. Let's say you want to multiply "x" by "y" which we defined above. Let's see what happens

```
x*y
```

We see that R has multiplied "x" which is 2 by each number is object "y" the string of numbers 1 through 5. This is because R uses matrix math for all its operations.

Now let's change "x" to be an object that has 2 variables and multiply like we did before

```
x <- c(1,2)
x*y
```

And oh no!!, We got an error message saying “x” and “y” are not the same length. In R, things have to match for everything to work together. Always be sure to read the **error messages**. A lot of the time they are really helpful and can aid in debugging and solving your issue.
But let’s change “x” so that is matches the length of “y” and multiply like before:

```
x <- c(6,7,8,9,10)
x*y
```

Now we get a vector (a list) of numbers that have been multiplied in the order you gave them.

<u>Matrices</u>

R is super useful when dealing with a matrix. A **matrix** is 2 dimensional and is composed of **rows** and **columns** of **numbers**, in that order. We can make a matrix by combining 2 vectors (like we were using above). There are 2 easy ways to combine vectors in R, rbind and cbind. rbind attaches things along a row while cbind attaches things along a column. Let’s try each one with the variables we used before:

```
rbind(x,y)
```

what do we get? It’s a matrix made up of 2 rows and 5 columns. Now try cbind:

```
cbind(x,y)
```

Now we get something that has 5 rows and 2 columns. Let’s work with this one for now. In R each time you do an operation it will NOT save until you give that operation an **object** name so let’s name this dat

```
dat <- cbind(x,y)
```

So what does dat look like, call it by its name by typing in “dat”. Another useful command is dim( ). This dim command stands for dimensions and will give you the dimensions of your object.

```
dim(dat)
```

And remember, R works as rows and columns so this matrix as 5 rows and 2 columns.

```
Exercise 1: Now create another matrix called “hat” that has the
same dimension but different numbers as dat.  Then multiply the
2 matrices dat and hat.  What do you get? How is R multiplying
this?
```

Data Frame

Data frames are a lot like matrices in R except that a matrix only contain numerical data. A data frame can contain numeric data, categorical data (like species name), and logical expressions (TRUE or FALSE). Much like a matrix, a data frame has 2 dimensions: rows and columns in that order.

Let's first create a data frame by using what we have already built. Combine the matrix called dat with the matrix called hat by column. Remember to name it something or R will not remember it:

```
cbind(dat, hat)
dat2 <- cbind(dat, hat)
```

Now let's look at it by calling the new object you just made:

```
dat2
```

Now, let's add a new column that has categorical data and let's call it sp for species. These will be fake species. Since we have 5 rows we will need 5 species

```
sp <- c( "a", "b", "c", "d", "e")
dat2 <- cbind(dat2, sp)
dat2
```

But see how everything is in quotation marks, that's not good, R can't figure out what is going on so we have to tell it it's a data frame with a really simple command

```
dat2 <- as.data.frame(dat2)
```

Now the matrix is a data frame. But how do we know? Here is a handy command to tell you what you are working with. The command class( ) tells you what the class or type of object you are working with is:

```
class(dat2)
class(hat)
class(x)
```

Now that we have a data frame let's make it more practical and name to columns like a real data set. This can be done in one command in R called names ( ) which provides names to the columns of a data frame.

```
names(dat2) <- c("footL", "ID", "temp", "tailL", "sp")
```

Now our data frame has real names! Let's say you are curious about the values of certain species, you can imagine this would be a pain if your data frame was HUGE. There are great short cuts in R to do this. For example, I want to know the foot length of species "d". We can pull out that value

```
dat2[[4,1]]
```

The brackets mean pull out, double brackets are usually used for a single element and single brackets do the same thing but general refer to a list of elements. Remember, data frames are rows and columns, that's why it was 4,1 and not 1,4.

Now let's say you want to pull out only the foot lengths because you are curious about them, there is a special **operator** that pulls values out of data frames and that's the **dollar sign $**. With the dollar sign operator, you have to use the column name:

```
dat2$footL
```

The dollar sign is telling R to pull out all the elements in the column called footL.

Plotting in R

R is also really great for making things from scatter plots to phylogenies. Let's plot foot length and tail length to see if there is a relationship between the two. The command for plot is….plot. The plot command requires two things, and x and a y. It will always be in that order, x then y. So let's see if we can plot

```
plot(dat2$footL, dat2$tailL)
```

What a weird graph, that is not what we wanted. What is going on here? When we made the data frame earlier in our code, R tried to figure out what each column was and it does not recognize our numbers to be numbers. This is an easy fix and happens often in data frames. First, we want to know what the class of the columns are:

```
class(dat2$footL)
class(dat2$tailL)
```

This is not the format we want, we want those columns to be numeric. Now let's switch the values that should be numeric into a numeric format, this is also straight forward. R gets confused sometimes so you will have to make it into a character first then a number but this can be done in one piece.

```
dat2$footL <- as.numeric(as.charater(dat2$footL))
dat2$tailL <- as.numeric(as.character(dat2$tailL))
```

```
dat2$temp <- as.numeric(as.character(dat2$temp))
```

So what we did there was make our column numeric by using the command as.numeric then saving back into the same column, essentially we replaces the old column with the new one with the correct class. Now if we look at the class of these columns is should be numeric:

```
class(dat2$footL)
class(dat2$tailL)
```

Now let's try to plot again using the exact same command. One great thing about R is that you can call an old command by pressing the "up arrow" on your keyboard. This will go through all the previous commands in order. You can also just type it again:

```
plot(dat2$footL, dat2$tailL)
```

So we see that as a foot get larger the tail also gets longer. The function plot has a lot of really fun arguments that you can play with to make plots pretty. Be careful not to waste too much time here, it's easy to get lost playing with colors:

```
plot(dat2$footL, dat2$tailL, pch=16, cex=2, col="red")
```

So what are all these arguments, pch = is shape of point and there are all kinds of shapes, cex is the size of the point, and col is the color of the points. These are the most basic things but you can get really complicated. The default here is scatter plot because both x and y were numeric variables, but let's say we want to plot the tail length of each species

```
plot(dat2$sp, dat2$tailL)
```

That's ugly, let's make it a bar plot

```
barplot(dat2$tailL, names=dat2$sp)
```

barplot is a function that forces a plot into a histogram. You give it the height, which will be the vector of y values and the names argument give it the x-axis categories. Now let's label the graph so people know what each axis is

```
plot(dat2$tailL,  names=dat2$sp  xlab="species", ylab="Tail
          Length", main="Tail lengths of 5 species")
```

"xlab" is the x-label, "ylab" is the y-label, and "main" is the title of the graph

Question: Why are we putting some things in quotation marks and other things don't have anything around them?

Directories

Directories, like in any programming language, are super important. These are **paths** that tell your computer where to get things from, where to save things, and where it currently is, it's the GPS of computer languages. In R, one of the first things you should do is set the directory to tell the computer where to look and save files. This is done by the command:

```
setwd(" my directory ")
```

this stands for **set working directory**. Where is my directory? In macs, the easiest thing to do is just to create a new folder that you will be working in and drag the folder in between the quotation marks. On a PC, open the folder you want and on the search bar, just copy and paste that directory in between the quotes. This will be different for EVEYONE because everyone is using a different computer.

If you think you don't remember setting the directory but aren't sure, using this command will tell you where you are

```
getwd()
```

`Exercise 2: Everyone make a new folder and set your directory to that folder and make sure it is correct.`

Reading in files

Now that we know a little bit about how R works we can get started on some real life data. I'm providing the morphological data I collected for my dissertation. R can read in many types of files but the most commonly used is probably spreadsheets. These contain data collected in the field, in the lab, or from videos.

I think the best way to save spreadsheets is in the csv format (comma separated). This is the easiest thing for R to read in. You can save files in this format from excel.

Let's get started, let's read in my morphological data

```
read.csv("morphdat.csv")
```

Well, that doesn't look nice! See how each value is separated by a semi-colon and not a comma, R is expecting a comma so it doesn't know what to do. This is an easy fix by telling R to separate thing by a semi-colon. We can do this by using the argument **sep** and telling it what to look for.

```
read.csv("morphdat.csv", sep=";")
```

But remember, this just reads in the file, it doesn't save it, you have to give it a name

```
morph <- read.csv("morphdat.csv", sep=";")
```

Now we can see the entire spreadsheet by calling the object called morph. But what if we just want to make sure it got read correctly without going through the entire thing? R has two commands that can do this

```
head(morph)
tail(morph)
```

These two pieces allow you to see the first and last 6 lines so make sure the entire thing was read in. It's also useful if you forget the column names and don't want to load the entire thing.

My spreadsheet contains several things going from left to right: row number (that is useless), JR numbers (my personal ID), IDs (field IDs), genus, species, gensp (genus and species together), sex, snout-vent length (SVL), femur length, tibiofibular length, tarsus length, foot length, head length, head width, humerus length, radioulna length, hand length, finger pad length, finger pad width, toe pad length, toe pad width, finger area, toe area, interdigital webbing, and ecomorph.

My dissertation involved describing frog ecomorphs via morphological specialization to disparate habitats. My hypothesis was that species will be more similar in morphology if they share the same ecology rather than because they are closely related. There will also be specialization in morphology to navigate their different habitats.

First, let's clean up the data by taking that first column called "X" out. This is easy in R

```
morph <- morph[ ,-1]
```

Here, we are using the single bracket because we don't want just one element, we want many. And remember how R reads things in as rows and columns, by leaving the row value (before the comma) blank it means "pick all rows". Then I added a -1 which means take out the first column. So the whole command is remove the first column and all the rows from that column. If you wanted to take out rows and not columns, fill the row numbers before the comma and leave space after the comma blank. You can also remove rows and columns together by filling in both spaces.

```
Exercise 3: Remove rows 3 and column 6, but don't save it.  Next
remove rows 3,12, 325 and columns 1, 4, 5.  Don't save any of
this, it's just to try it out.

Now instead of deleting, pull out rows 3, 12, 325 and columns
1,4,5.
```

<u>Data exploration</u>

Data exploration is super important in science, it helps you get familiar with your data and identify errors that you might have missed. This usually involves plotting things that you are interested in just to see the spread of the data.

For example, my frogs have a size range that's over an order of magnitude big. Let's plot the SVL by gensp.

```
plot(morph$gensp, morph$svl)
```

Let's add labels to the graph, but the names are also hard to read, the las=2 command makes the names go up and down and not side to side

```
plot(morph$gensp, morph$svl, xlab="species", ylab="mm", las=2)
```

These boxplots are great because we can see if there are outliers or something crazy that doesn't make sense. I don't see anything too weird so we can assume it's good to go.

Now let's say that we are interested in the morphological differences between the ecomorphs. ***As an aside, ecomorphs are a set of closely related species that show morphological specialization to different habitats and are better equipped to perform certain tasks when compared to other species of different ecomorphs. The most famous example is Darwin's Finches. They evolved different shaped beaks that are specialized to eat particular food types and perform poorly when eating different kinds of foods. The frogs possess a similar story but instead of beaks, we will be looking at limbs. ***

```
Exercise 4: Plot to different limb elements by ecomorph to get a
feel about how different these frogs are. Try plotting in
different colors and see what you can do. You can use the
command colors( ) to see all the colors available in R.
```

Now that we see how different species have different sized limbs we have to correct for size. This is very common when working with animals because animals vary tremendously in size (mouse vs blue whale). In order to compare across different sized animals, we correct for size to see how variation really is spread. In our case, we will divide all limb elements by snout-to-vent length (SVL) so that limbs are now proportional to the size of the animal. That way we can see if limbs are longer or shorter than they should be for the size of the animal.

```
morph$Sfemur <- morph$femur/morph$svl
```

So what is going on here? Basically on the right size we have the femur column being divided by SVL for each and all rows, R does the matching automatically. Then we are making a new column called Sfemur (for size-corrected femur) and putting into the morph data frame. Notice that Sfemur did not exist before but you can create a new one just by naming a new column in your data frame and putting something in it.

Now do it for all the hindlimb measurements (femur, tibiofibula, tarsus, foot)

```
   morph$Stib <- morph$tib/morph$svl
morph$Starsus <- morph$tarsus/morph$svl
  morph$Sfoot <- morph$foot/morph$svl
```

Great, now that we have all the data that we need we can start with some simple statistics. This next section is not a stats crash course and won't be as in depth as a real stats class. But after this, you should be able to perform simple statistical analyses that are very common in biology.

Statistics

Hopefully, everyone has taken a stats course. If you have not, this will go over a few things but it is very introductory and we will not be covering all of stats. If possible, I highly recommend you take a stats course; it will make your life easier. In most non-medical sciences, we use a p-value of 0.05 to indicate differences between variables or treatments. In medicine, the cut-off is stricter, 0.01, since they are dealing with humans. There are two types of data: **continuous** and **categorical** data. Continuous data are numbers that can be measure on a continuum scale, for example measuring you finger length. Categorical data is data that fits into a category, like eye color can be blue, or brown, or green. We will be using both these type of data in our analyses.

First, we will run a **t-test**. This compares two sets of continuous variables to see if there is a difference. Let us say that we are interested in knowing if femurs are different lengths compared to tibiofibulae. The command for a t-test is t.test( ). Shocking, I know. It requires two arguments which will be the variables we are comparing separated by a comma:

```
t.test(morph$Sfemur, morph$Stib)
```

R should return output which is the results from the test. We have a t-value which measures the size of the difference relative to the variation in your sample data. We have df which are degrees of freedom of your samples. Degrees of freedom is the number of values in the final calculation of a statistic that are free to vary. Lastly, we have our p-value which is 0.004754. This is less than 0.05 so we can say that there is a statistical difference between the femur and tibiofibula. If it was more than 0.05, let's say 0.07, we would say there is no difference.

```
Exercise 5: Just for fun, compare the size-corrected hindlimb
morphology to see if there are differences, they should all be
different.
```

t-tests are very useful when comparing two continuous variables but what if we want to compare many variables at once. We need to run an **ANOVA** which stands for Analysis Of Variance. ANOVAs require both continuous data and factors which we can compare across. It will test to see if the variance between 2 or more groups is different, if it is, we can designate the result as statistically different. ANOVAs run with formulas that use **a tilde (~).** The tilde (located next to the number 1 on your keyboard) means as a function so that x~y is read x is a function of y. So that x changes in response to changes in y. So let's say we want to know if femur length varies across the ecomorphs we would write:

```
aov(morph$Sfemur ~ morph$ecomorph)
```

The output isn't as useful as the t-test. It is mostly talking about residuals because the test is based on residuals but what if you want more info. You have to make the test an **object**

```
femur.aov <- aov(morph$Sfemur ~ morph$ecomorph)
```

Then we can look at the summary of the test by using the summary command

```
summary(femur.aov)
```

Now the output looks more familiar. Make sure you look up what everything means like **Sum Sq** and **mean Sq** but if we look at the p-value (Pr>F) it is really small! **Note, P-values are not everything, you have to understand the whole output to understand the story*** This means that across ecomorph category, the femurs are not the same length, even when corrected for size. But that is all the information we have, how are they different? Are they all different? That information is not conveyed so we have to perform one more *post-hoc* test called the **Tukey test**. This will tell you the specifics of everything

```
TukeyHSD(femur.aov)
```

The output is a data frame that gives you which categories are being compared, the mean difference between the two, lower and upper bounds, and the p-value. You see in the p-value column that not everything is different! Some are and some are not, but R picks the lowest number to report so although the ANOVA might say there is a difference it is important to investigate what and how variables are different. You do not want to convey incorrect results.

Linear regressions is a statistical test that allows you to model the relationship between a scalar response (more commonly known as the **dependent variable**) and an explanatory variable (**independent variable**). Much like an ANOVA, it uses the tilde (~) to represent the relationship between the two variables you are trying to model. For example, let's say you want to know if as the femur in the leg of a frog gets longer, so does the foot. Both variables have to be continuous for the linear regression to run.

```
lm(morph$Sfoot ~ morph$Sfemur)
```

The output only provides the coefficients and intercepts. But what is we want more information? We can save the output as a summary like we did with the ANOVA.

```
foot.lm <- lm(morph$Sfoot ~ morph$Sfemur)
```

And last, we can check out the summary

```
summary(foot.lm)
```

Here, we get a summary of the residuals, the important coefficients like standard error, t-value, and p-value. You also get your F-statistic and R-squared. In linear models the R-squared is very important because it tells you how well your model is doing. Basically, what proportion of the variance for the dependent variable is explained by the independent variable. In our case, the R-squared is .30.

We can also make a plot of our regression so we can see what our model looks like. First we start of by plotting a x,y scatter plot. Remember that in this example, the femur is the independent variable and the foot is the dependent variable:

```
plot(morph$Sfemur, morph$Sfoot)
```

Now we want to plot the model that we created. We do this by using the command **abline( )**. This command creates a straight line guided by single values specified by “a” which is the intercept and “b” which is the slope (hence, abline). In order for this work, the scatter plot you just created has to be opened.

```
abline(lm(morph$Sfoot ~ morph$Sfemur))
```

Now we can see our model in a 2 dimensional plot.

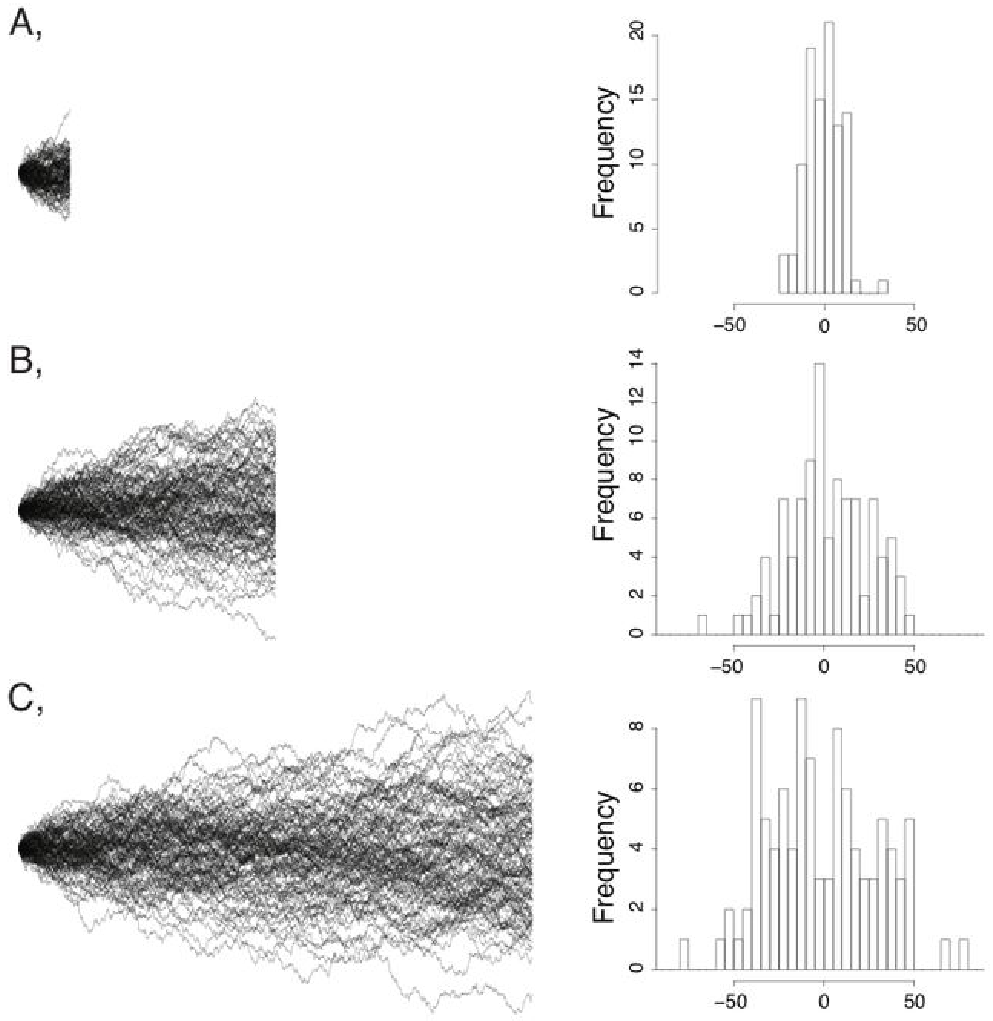
Comparative methods

We use comparative methods when studying evolution to correct for the fact that all of life is related. Because all species are related to some degree, you cannot use normal parametric statistics because data points are not independent (due to phylogeny) and violate parametric assumptions. There are several models and processes that can explain evolutionary patterns that we see in nature. Here we will go through two major processes used in comparative methods, Brownian motion and Ornstein-Uhlenbeck process.

We will start with the simplest model: Brownian motion (BM). The process is derived from physics, which defines it as "the erratic random movement of microscopic particles in a fluid". In biology, we take that definition and change it a bit: "the evolution of characters are a function of neutral evolution due to genetic drift" (Lande 1976). The assumption is that a character is influenced by many genes, each having a small effect, and that the value of the character does not affect fitness. Because there is no selection, the phenotypic character will change only due to mutations and genetic drift. BM is an example of a "random walk" over continuous time because the trait value can change randomly in both directions and distance, over any time interval. BM can be defined using the following equation:

$$\sigma dB(t)$$

where B(t) is the standard Wiener process (Brownian motion) and σ is a matric quantifying random drift. This is a simple model and we can see how a trait might evolve through time below:



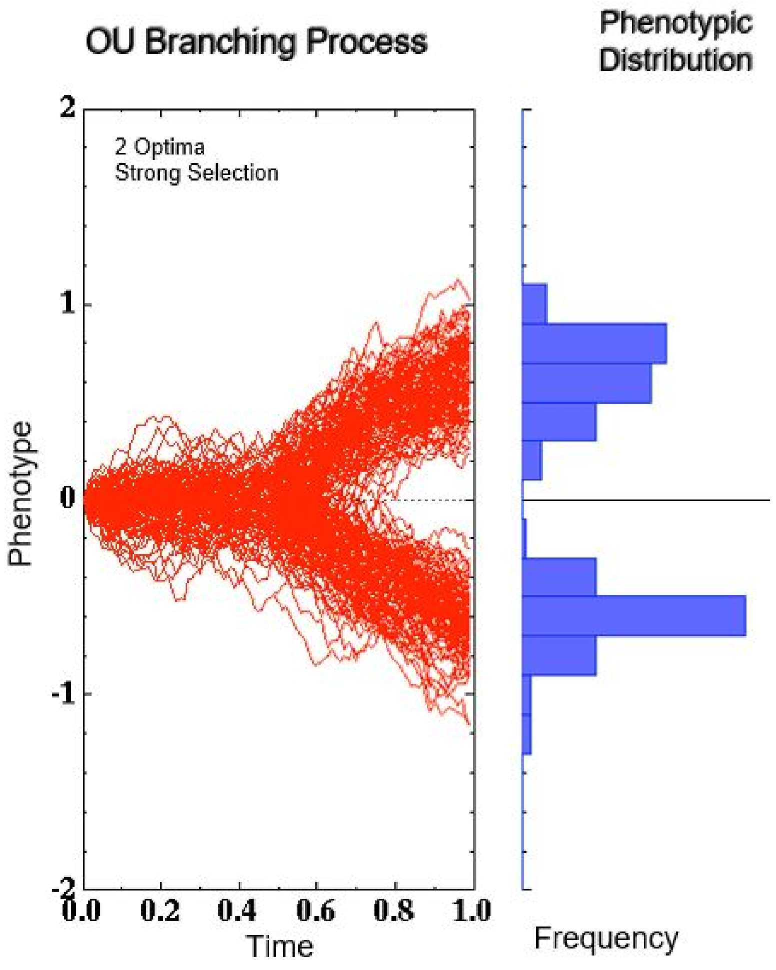
Examples of Brownian motion. Each plot shows 100 replicates of simulated Brownian motion with a common starting value and the same rate parameter $\sigma^2 = 1$. Simulations were run for three different times: (A) 10, (B) 50, and (C) 100 time units. The right-hand column shows a histogram of the distribution of ending values for each set of 100 simulations.

As we can see, BM leads to a diffusion process that creates a Gaussian distribution of the phenotype. The longer you let the BM process run the wider the distribution of phenotypes will be but they will still follow the normal bell shaped curve. Although the distribution might get wider (or narrower), there isn't a shift in the mean so the mean trait value remains that same.

On the other hand, the Ornstein-Uhlenbeck process (OU) describes the evolution of a trait under a model of selection rather than just a random walk like BM. The OU process was derived from physics and it was meant to explain for velocity of a massive Brownian particle under the influence of friction. In biology, we use to it explain a trait that is under selection. We use the following equation:

$$dX=\alpha(\theta(t)-X(t))dt+\sigma dB(t)$$

Where your α is the strength of selection, your θ is your optimum phenotype value X is multivariate trait value along a lineage. Again, α is a matrix quantifying the strength of selection much like σ in the stochastic part of the equation. How this works (very simply) is that as the strength of selection increases, your trait will reach its optimum fast. If the strength of selection is small, it will take a long time to reach the optimum.



Example of the OU process with strong selection. The traits diverge at time 0.5 and evolve to reach 2 different optima which each have a Gaussian distribution that are different from one another. This is an example if a trait that has diverged and to populations (or species) have different trait mean values.

This was a very crude and quick explanation but there are many resources out there to explain things in more details. Now we will actually implement this in a dataset that is published. We will run an OUCH and SLOUCH analyses. This are similar but SLOUCH has the benefit of comparing continuous and continuous data (regression) while OUCH can only compare continuous to categorical data (ANOVA). You can also include standard error in your SLOUCH analyses to provide more informative data.

Before we being modeling, we will use simple parametric tests, like the ones above, but corrected for phylogeny. Remember to follow along with the code provided. I will explain things in this document and the script will help you with the code.

```
dat <- read.table("12spJND.txt", header=TRUE)
```

First we will read in the data with the values that we need. These is reflectance data from the dorsal and ventral parts of Sceloporus lizards. JND are just noticeable differences, which is a measure of how much they stand out against the background. In Sceloporus lizards, there are species that are blue-bellied species and white-bellied species. We want to see if these species are different and what is driving the differences. Here, we meaure 4 traits ventral JND (vJND), ventral achroma (vAchr), dorsal JND (dJND), and dorsal achroma (dAchr).

Then we will read in the tree and make it ultrametric so that all branches end at present time

```
tree <- read.tree("12sptree.tre")
tree <- chronopl(tree, lambda=1)
```

Then we will take the means by species and sex because lizard coloration is dimorphic in Sceloporus. This means that males and females look different so we can’t pool that data. We will also remove the unused columns from our data set and rename everything so we know what everything is.

```
dat2 <- aggregate(dat, by=list(dat$species, dat$sex), mean, na.rm=TRUE)
dat2 <- dat2[,-c(3,6,7)]
names(dat2) <- c("species", "sex", "VentJND", "VentAchrom", "DorsJND", "DorsAchrom")
```

Then we will add the ventral coloration so each species is designated as white or blue. We could do this manually but I wrote some code to make things easier on us.

```
blue <- c("parvus", "grammicus", "occidentalis", "undulatus", "merriami", "jarrovii", +
"graciosus", "variabilis")
white <- c("siniferus", "cozumelae", "megalepidurus", "virgatus")

blueoo <- dat2$species %in% blue
whiteoo <- dat2$species %in% white

dat2$belly[blueoo] <- "blue"
dat2$belly[whiteoo] <- "white"
```

Lastly, we want to split the data frames by sex, we are only interested in males, and make sure that the name of each row is the species name and not a number.

```
datM <- dat2[c(13:24),]

dat2M <- datM[,-1]
rownames(dat2M) <- datM[,1]
```

First, we are going to perform a phylogenetic ANOVA to see if we can detect differences. We need to make each trait its own variable with the species names associated with the species means. Remember to make the belly color a factor.

```
dat2M$belly <- as.factor(dat2M$belly)
vJND <- dat2M$VentJND
names(vJND)=rownames(dat2M)

vAchr <- dat2M$VentAchrom
names(vAchr)=rownames(dat2M)

dJND <- dat2M$DorsJND
names(dJND)=rownames(dat2M)

dAchr <- dat2M$DorsAchrom
names(dAchr)=rownames(dat2M)
```

We also have to associate the color with each species, same procedure as we did above but just with belly color now.

```
belly <- dat2M$belly
names(belly)=rownames(dat2M)
belly <- as.factor(belly)
```

The last thing we should check is the phylogenetic signal. This tells you if the trait you are studying is different because of phylogenetic history of because of something else.

```
phylosig(tree, vJND, nsim=1000, method="lambda")
```

Lambda here is modified from Pagel's lambda and ranges between 0-1. 0 means no phylogenetic signal (the pattern is being driven by selection or stochastic events) and a value of 1 means that the pattern in your data is driven by relatedness and not selection or drift.

Phew, now we can run the pANOVA using the following code:

```
MvJND <- aov.phylo(vJND~belly, tree, nsim=1000, test=c("Wilks", "Pillai"))
summary(MvJND)
```

Remember that you will need the variable you are testing (vJND), the categorical variable (belly) and a tree (tree). We will run this analysis 1000 times and have a few tests associated with it. Remember **THE TREE NAMES AND DATA NAMES HAVE TO MATCH EXACTLY** for this work. So what do we get? We don't find a difference between blue and while vJND and our R-squared really doesn't tell us much. So this is telling us that in this moment in time in evolutionary history, we cannot detect differences in male ventral JND between blue and white species. This does not mean that there isn't a historical difference. For that we have to model using OUCH and SLOUCH. We will now run those analyses.

Again, there is a lot of data processing to get to the point that we can run the analyses. First we have to create a data frame from our phylogeny and populate it with our data and our hypotheses. This is a really important step because this is where you will explicitly test hypotheses of evolution so they should be realistic and biologically meaningful. I have already constructed this for you so I won't go into how it is done but the code on how to do it is in the script.

We will first read in our new tree data frame with hypotheses and data. Then we give each row the node name so the tree can match the data with the right node. Then we can plot the hypotheses one by one to see what they look like. The ancestral reconstruction is VERY important. You might want to find another function to help you do it. If the ancestral reconstruction of you hypotheses is bad, you will get bad results.

```
tdat <- read.csv("treedatframe.csv") #read in tree data frame
rownames(tdat) <- tdat$nodes
tree <- ouchtree(tdat$nodes, tdat$ancestors, tdat$times, tdat$labels)
```

Now let's plot all the hypotheses one by one.

```
plot(tree)
plot(tree, regimes=tdat["hab"], lwd=5) #habitat type color regimes
plot(tree, regimes=tdat["elev"], lwd=5) # elevation
plot(tree, regime=tdat["belly"], lwd=5, cex=.75) #belly
plot(tree, regime=tdat["OU1"], lwd=5, cex=.75) #global optimum
```

Next, like we did with the ANOVA, we want to create factors for our hypotheses and data so we can test them

```
#make OU1 hypothesis of global optima
ou1 <- factor(tdat$OU1)
names(ou1) <- tdat$nodes
```

```
#making variables into factors for analysis
habitat <- factor(tdat$hab)
names(habitat) <- tdat$nodes

elevation <- factor(tdat$elev)
names(elevation) <- tdat$nodes

belly <- factor(tdat$belly)
names(belly) <- tdat$nodes
```

OK, as you can see, it takes a long time to get to the point where we can even start the analyses. You have to make sure everything is correct and as it should be or, as you know, garbage in garbage out. It takes a long time to set these analyses up so make sure you put in the time in the front end so you aren't regretting it at the other end.

The way OUCH works is that you feed it the trait you are interested in, the phylogeny, the hypotheses, and a start value for the alpha and sigma matrices. The start value for alpha and sigma are arbitrary but starting them at 1 is usually a good place to start.

```
#this takes the data from your tree data frame with the nodes associated with them
vJND <- tdat$VentJND
names(vJND) <- tdat$nodes

#Then we make an empty list so we can populate it with the analyses.
H.vJND <- list()

#Then we test our hypotheses

#this one is brownian motion, remember, there is to hypotheses, everything is random
H.vJND$BM <- brown(vJND, tree)

#the code below tests the differnt hypothese of habitat, elevation, belly....
H.vJND$hab <- hansen(vJND, tree, habitat, sqrt.alpha=1, sigma=1)
H.vJND$ele <- hansen(vJND, tree, elevation, sqrt.alpha=1, sigma=1)
H.vJND$belly <- hansen(vJND, tree, belly, sqrt.alpha=1, sigma=1)
H.vJND$OU1 <- hansen(vJND, tree, OU1, sqrt.alpha=1, sigma=1)

#then we make a table that has all the information we need from all the different hypotheses we tested.
Hfit.vJND <- t(sapply(H.vJND, function(x) {as.data.frame(rbind(unlist(summary(x)[c("dof",
"loglik", "deviance", "aic", "aic.c", "sic")])))}))
```

```
#and we print the results
print(Hfit.vJND, digits=3)
```

Because this is a small sampling of all of Sceloporus, we want to look at the AIC.c which is AIC corrected for small sample sizes.

From our results, we can see that Brownian motion best explains ventral JND differences. This isn't so surprising since we didn't find differences in the phylogenetic ANOVA. Now, let say one of the other hypotheses best fit the data, like our habitat hypothesis. We can check the output by calling from our list.

```
H.vJND$hab
```

You will notice that you get the tree data frame. Under that you get alpha, the strength of selection, and sigma, the noise. You can see that from our data, there is a lot of noise which could be why this model isn't the best fitting. The important thing about OUCH are the thetas. There are the optimum value for each factor in your hypotheses. What this is saying is that the optimal value for arboreal species is a JND of 30.13, for terrestrial is 17.95 and so on. These values will change a bit as your run them each time because you are modeling it. The cool thing is to see if each factor has a different optimum and this can be informative. Maybe arboreal species have the highest contrast for some reason while saxicolous species blend in the most. Then we can start thinking about why that could have evolved. Again, BM was the best model so we cannot infer much but you can see how cool this would be if another model would have performed the best.

Now we will run a SLOUCH analysis to see if we get similar results and learn how to interpret the output. You will need to install SLOUCH from a GitHub and I have the instructions on how to do it in the code. Again, OUCH and SLOUCH are very similar, but, the interpretation of the values are a bit different. In OUCH, the alpha and sigma are uninterpretable alone and are only informative in reference to each other. In SLOUCH, you can actually interpret these variables and give each one of them meaning. Also, you can incorporate errors into SLOUCH to get more precise modeling. I have done this already for this tutorial. Unfortunately, SLOUCH is coded in such a way that you need to code it EXACTLY as I have it or it won't run. For this section, I won't write out the details here but in the script to make things easier. We can check back here for the interpretation of results.

After you run your SLOUCH analysis you will get an output with several variables. The Rate of adaptation is similar to your strength of selection. It measures how strongly selection is acting on your trait. The phylogenetic half-life is the inverse of rate of adaptation (when RoA is large) phylogenetic HL is small. This is a measure of how long it will take your trait to reach its optimal value. You also get the thetas that OUCH gave you but here they are called optimums. You get one for the ancestral (0), one for the first trait, in our case blue coloration (1) and the ancestor (A). You also get your model fits so you can compare them to other hypotheses. The other really cool thing about SLOUCH is that you get to visualize your phenotypic space. In our example, you can see there is an optimum value that the trait is evolving towards. You can compare this fit with one of Brownian motion to see which one best fits your data.

I know this is a lot to take in but you will get used to it. Just keep playing with the data but remember that the most important thing when it comes to comparative methods is that your hypotheses are REALISTIC and biologically meaningful. Of course you can find a model that fits your data perfectly, but would that actually occur in nature? As biologists, we are trying to find patterns in life so as try to explore these, we have to keep in mind what are realistic assumptions and what are assumptions that are mathematical and maybe do not apply in the real world.