Task 05: Revisiting a Lab

I. General

Type up all of your work in a text editor. Basically, you should NEVER type things directly into the `R` terminal. Type them into a text editor, then either run them or copy/paste them into `R`.

Before you begin, make a new folder in Tasks called Task_05, and save an empty file named task03.r in that folder.

When you're **done** with this assignment, turn it in by (1) saving your text document, (2) opening your Terminal or GitBash, (3) navigating to the appropriate directory using `cd` and (4) typing:

`git add -A` (enter)
`git commit -m "yourname Task 04"` (enter)
`git push -u origin master` (enter)

II. Background

This `R` activity has three separate parts. First, you're going to do some simple simulation experiments using the `simPop` function. These are called *in silico* experiments, as they are done on a computer. You'll make a (virtual) population of little hermaphroditic creatures that mate and then die (non-overlapping generations; individuals can mate more than once though!). You'll track their fitnesses and look at how allele frequencies change as a function of population size, heritability of allele b, and selection acting on allele b.

Next, you'll get a touch more realistic. Instead of experimenting on simulated populations, you'll be reanalyzing data from a BIOL 112 lab in which students simulated evolution physically by "eating" asexually reproducing sticks. Unlike the simulation, you do not know the relative strength of selection here (as it was produced by students), and you cannot vary the population size (the experiment is over, these are just the data).

Finally, you'll combine the two approaches. You'll simulate additional trials of the student-led experiment, and compare your simulated results to the observed results in an attempt to learn something about how students behave durin this lab.

III. Simulation

We're going to be doing simulations in this handout. We touched on them briefly last week, but this week is going to be heavily driven by them. A simulation is when, on your computer, you invent a fake world with a set of rules that you decide on. Then you let that world go and grow for a period of time (defined in those fixed rules), and have it tell you how things went.

The `simPop` function we're going to use invents a world of hermaphrodites that have exactly two alleles and reproduce sexually, but whose population size never varies (that is, each generation

there are always the same number of individuals). This population progresses from some start point for some number of generations, and then you receive a record of the frequency of both the two alleles and all three possible genotypes for each generation.

You can control the simulation by setting the arguments in the `simPop` function to different values. Adjusting the `Popsize` argument changes the population size, for instance, while the `nGenerations` argument sets how long the simulated world should run. `initial_p` controls the starting frequency of the a allele, while `h` and `s` control the heritability and selection coefficient for the b allele.

As with last week, this handout mostly just shows you the basic usage of `simPop`, but as you can see, you can use it to study. I strongly encourage you to do so. Look at how effective selection is when the allele is recessive versus overdominant, look at how drift and selection interact. Does the most fit allele always go to fixation? What combinations of drift and heritability prevent that? There are many nuances to our material that you can explore with these functions. I've also provided a convenient plotting tool.

```
source("http://jonsmitchell.com/code/fxn05.R")

# run a simple simulation
Pop1 <- simPop(Popsize = 50, nGenerations = 100, initial_p = 0.5, h =
    1, s = 0)

plot(1:nrow(Pop1), Pop1[,1], ylim=c(0, 1), type = "l", xlab="generation
    ", ylab="allele freq.", lwd=2)
lines(1:nrow(Pop1), Pop1[,2], lwd=2, col='red')
legend("topleft", legend = c("a", "b"), col = c("black", "red"), lwd =
    2, bty="n")

# I also set up a plotting wrapper function. This works mostly the same
    as normal simPop
# except you can set the number of runs (nruns) to do multiple
    simulations at once.
# It will plot the relative fitnesses of the three genotypes (left) and
    the frequencies of
# allele a for each of the nruns populations with a different color per
    simulation

plotFit( nruns = 10, n = 50, ngens = 100, init_p = 0.5, h = 1, s = 0 )
```

IV. Days Long Past

This week, we're going to return to a lab you did in the past if you took BIOL 112 here at Tech. The third lab in that class was a Natural Selection experiment involving toothpicks. If you didn't take 112 here, or if you just want a memory refresher, students formed groups and spread 60 colored toothpicks (10 toothpicks of each of 6 colors) onto a colored paper background. One student "ate" the toothpicks one at a time until only 20 toothpicks remained. Then, the 20 "surviving" toothpicks "reproduced" to create two additional copies of their color back into the pool, resulting in 60 toothpicks again. Predation and reproduction were then repeated twice. At the end, the χ^2 statistic was calculated ($\frac{(expected-observed)^2}{expected} = \chi^2$).

For reference, the 112 lab handout is posted on eCampus for you to re-read!

The datasheet you'll be analyzing is the "whole class" data file from the 2012 to 2020 labs combined. Each row represents the final population size for each of the six colors, for one group, on one background, from one year. In that freshmen-level class, you learned a lot of foundational things (that you must remember for this class! It's a prerequisite!) but, as it is a freshmen-level class, there were some necessary simplifications. We're going to explore one of those by reanalyzing the data from that lab.

V. What is a test statistic?

A *test statistic* is a way to quantify how different a set of observations is from your expectation. For instance, in a t-Test, you compare the difference in the means between two groups (this is the "t statistic"). You *expect* the t-statistic to be zero if there's no difference between groups. But random variation in samples means that the test statistic may not be exactly zero even if there is no meaningful difference between groups. So the further from zero your observed value is, *clearer it is* that the two groups are different.

Almost all "statistical tests" you know/at one point learned are based on this idea. You calculate some measurement of your data relative to some expectation (usually zero) and get a number that describes how far your observed data are from that expectation. Then you compare your calculated number to a known distribution (usually in the form of a table) to get the "p-value". The notion is that the p-value tells you the probability your data differ from your expectation due solely to sampling errors. However, this is essentially never true, as p-values are accurate if and only if all of the assumptions of your test statistic are met and your expectation isn't unwarranted.

If the above paragraph seems confusing (or if you've had a statistics class and found it confusing), don't fret: it seems confusing because it *is* confusing. The whole idea of using "standardized" test statistics only makes sense in a pre-computer world where you needed to look things up in published tables. Because you couldn't have an infinite number of published tables, people would only have a few (e.g., a table for the t, F, or χ^2-statistics) and then would design experiments *to fit with the tables they had.*

You can visualize this in `R`

```
# Let's assume that we expect to see four equal categories
Expectation <- c(10, 10, 10, 10)

# Now let's pretend what we actually see is four uneven categories
Observed <- c(15, 15, 5, 5)

# We can calculate the Chi-squared statistic using these numbers
Chisq <- sum( ( ( Expectation - Observed ) ^ 2 ) / Expectation )

# And we can visualize what this means pretty readily
barplot(rbind(Expectation, Observed), beside = T, main = bquote(chi^2 ~
    "=" ~.(Chisq)), legend.text=c("expected", "observed"))
```

Try setting the Observed vector to c(5, 0, 0, 35), and (2, 3, 10, 30), and other combinations that you can think of. What's the χ^2 value when you observe all 10's? What about when all 40 observations are only in one category? How does the χ^2 relate to the *evenness* of the bars?

Nowadays, we have computers. We can calculate tables trivially, if we want, or we can simulate whole alternate realities in which experiments were conducted differently, or assumptions were

violated and compare our observations to those alternate worlds.

This week's task is a little funny. We're going to try to learn about evolutionary processes by comparing how actual freshmen perform a physical experiment to how hypothetical perfectly-behaved freshmen perform a simulated experiment. We'll do this by looking at calculated χ^2 values from the 112 lab and comparing them to simulations.

We're going to take the observed data. We're going to calculate the test statistics and ask how often we find "significant" results. Then we're going to simulate an alternative scenario and compare our observed data to the simulation. We'll use that comparison to ask whether finding a "significant" result really means what we think it means, and see if we can figure out what evolutionary processes are most responsible for the observed changes.

To get started, open an empty .R file, as always, set your working directory, and begin adding in the following code:

```
# Read in the data from past years. I'd encourage you to look at this
    data. See Task 02a to remind yourself how to do that!
results <- read.csv("http://jonsmitchell.com/data/biol112labresults.csv
    ", stringsAsFactors=F)

# Simple subsetting of data to have only the right pieces
counts <- results[,c("yellow","red","green","blue","black","tan")]

# Let's list out the background colors (they differ from the toothpicks
    !)
backgrounds <- c("White","Red","Yellow","Green","Blue","Black")

# Let's set some (slightly nicer) than default colors. This line is
    optional.
backgroundCol <- c("white", "#d53e4f", "#fee08b","#abdda4","#3288bd","
    black")

# Now, let's calculate the Chi-squared statistic for the first row of
    counts.
calcChi(counts[1,])

# Now let's calculate Chi-squared for ALL of the rows at once!
Chisqs <- apply(counts, 1, calcChi)

# You've all seen Chi-squared values before, but interpretting what
    they actually mean, visualizing what the chi-squared number is
    telling you, might be a bit tricky. Use the following function
    multiple times.
plotChis(counts)

# Look at the plots. Look at the chi squared values for each panel, and
     the heights of the bars. How ``even'' are the bars when the chi
    squared is high? How ``even'' are they when it is low? What does
    plotChis() show you about how to interpret the number a Chi-squared
    test spits out?
```

```
173
174 # What's the average Chi-squared? How would you interpret that given
175     the (many) plots you should've just looked at?
176 Avg <- mean(Chisqs)
177
178 # How does the average chi-squared compare to the critical value in the
179     packet (11.70)?
180
181 # Does the chi-squared differ by background?
182 backgroundAvgs <- tapply(Chisqs, results[,3], mean)
183
184 # The critical value here is 11.70. That is, if the Chi-squared number
185     is greater than 11.70, then the p-value is <0.05.
186 # When the p-value is <0.05, we say that the difference between
187     observation and expectation is statistically clear
188 # So let's find out how many of our Chi-squareds show a clear
189     difference!
190 propSig <- length( which( Chisqs > 11.70) ) / length(Chisqs)
191 percSig <- round(100 * propSig)
192
193 # percSig tells you the percent of trials that had a ``significant'' p-
194     value. Does that number surprise you?
195 # As a bit of foreshadowing, do you think the only thing driving that
196     very high number is natural selection?
197
198 # Percents are good, but plots are better. Let's make a plot of Chi-
199     squared results.
200 par(las = 1, mar = c(4, 4, 1, 1), mgp = c(2, 0.5, 0), tck = -0.01, cex.
201     axis=1)
202 hist(Chisqs, main="", xlab="chi-squared values", ylab="frequency")
203
204 # Let's see if there're any background-specific patterns
205 # We'll set up an empty plot to put data into
206 par(las = 1, mar = c(4, 4, 1, 1), mgp = c(2, 0.5, 0), tck = -0.01, cex.
207     axis=1)
208 plot(1, 1, xlim=c(0, 400), ylim=c(1, 8.5), xlab="", ylab="", type="n",
209     yaxt="n")
210
211 # Because this plot is weird, we're going to setup the y-axis and x-
212     axis label separately
213 axis(2, at = 1:length(backgrounds), labels = backgrounds)
214 mtext(side=1, expression(chi^2), cex=1.75, line=2.5)
215
216 # Now, FOR each background, we'll add a histogram of the data.
217 counter <- 1
218 for (i in backgrounds) {
219   Data <- Chisqs[which(results[,3] == i)]
220   addHist(Y=counter, Dat=Data, Color=backgroundCol[counter])
```

```
  counter <- counter + 1
}

# Now let's add a line representing the critical value
abline( v = 11.70, lty=2, lwd = 2, col='black')

# The more to the right of that line a distribution is, the more often
    trials on that background were ``significant''
# Do you see any meaningful differences between backgrounds?
```

VI. Running a simulation

In the functions you downloaded from the website at the top, there's a function called `simDraws`. This function will simulate the toothpick experiment, but will do so "blindly". Each time, 40 of the 60 toothpicks are drawn *at random* without regard for color or background.

Basically, the `simDraws` simulates what would happen if the predator student had their eyes closed throughout the "eating" process.

```
# Let's simulate running this experiment without natural selection ten
    thousand times
Simulation <- simDraws(10000)

# Now let's add our Chi-squareds from the 10,000 simulations to our
    plot
addHist(Y=7, Dat=Simulation, Color="lightgray")
mtext(side=2, at=7, line=0, "simulated")

# And compare it to the line, too
abline(v = 11.70, lty=2, lwd=2)
```

Look at the relationship between the critical value and the simulated distribution. What percentage of the time does the selection-free simulation find a "significant" (greater than 11.70) result? Remember: a significant result means that there is a less than 5% chance that you'd get proportions as different as what you observed if you were just randomly drawing toothpicks. How can the selection-free simulation produce final counts so different from the initial counts?

Now, it'd also be interesting to simulate actual selection. That is, instead of simulating the procedure as if the student playing predator had their eyes closed, instead simulate the experiment as if the student playing predator *followed the instructions perfectly.*

The trick is, we don't actually know what that would look like! But just because we don't know what it would look like doesn't mean we can't simulate it. In fact, not knowing how it would work is actually where the power of simulations is greatest. We can simulate a *range* of scenarios, and compare those simulated scenarios to the observed data!

So that's what we'll do. The idea is that, on a given background, some toothpicks will be more likely to "survive" and "reproduce" than others. So we can set a "reproduce probability" (fitness; w) for each toothpick type.

```
# no fitness differences. The Fit value for each of the six toothpick
    colors are all 1.
Fit <- c(1, 1, 1, 1, 1, 1)
```

```
names(Fit) <- 1:6
Simulation2 <- simDraws(1e4, w = Fit)
addHist(Y=8, Dat=Simulation2, Color=rgb(0,0,0,0.25))

# one tooth pick type selected against
Fit <- c(0.1, 1, 1, 1, 1, 1)
names(Fit) <- 1:6
Simulation3 <- simDraws(1e4, w = Fit)
addHist(Y=8, Dat=Simulation3, Color=rgb(0,0,0,0.25))

# three tooth pick types selected against
Fit <- c(0.5, 0.6, 0.7, 1, 1, 1)
names(Fit) <- 1:6
Simulation4 <- simDraws(1e4, w = Fit)
addHist(Y=8, Dat=Simulation4, Color=rgb(0,0,0,0.25))

# five selected against
Fit <- c(0.1, 0.2, 0.3, 0.4, 0.5, 1)
names(Fit) <- 1:6
Simulation5 <- simDraws(1e4, w = Fit)
addHist(Y=8, Dat=Simulation5, Color=rgb(0,0,0,0.25))

# insane selection
Fit <- c(0.1, 0.1, 0.1, 0.1, 0.1, 1)
names(Fit) <- 1:6
Simulation6 <- simDraws(1e4, w = Fit)
addHist(Y=8, Dat=Simulation6, Color=rgb(0,0,0,0.25))
mtext(side=2, at=8, line=0, "sel. sim.")
```

Notice that none of these match the empirical distributions. The student-generated experiments have a peak that looks kind of like the no-selection simulations, but they also have a series of extra peaks out on the extreme end that look like the high-selection simulations. We can plot what a *mixture* of our simulations would look like really easily:

```
Simulation7 <- c(Simulation2, Simulation3, Simulation4, Simulation5,
    Simulation6)
addHist(Y=8, Dat=Simulation7, Color=rgb(0,0,1,0.25))
```

How does the mixture compare to the student-generated data? Looking at how the mixture was generated, do most student groups show evidence of strong selection? If you had to describe the relative strength of different evolutionary processes across all the groups in all the labs across all of the years, what would you say?

VII. Inference

What evolutionary process(es) are at work in the lab-as-done-by-humans?

What evolutionary process(es) are at work in the lab-as-simulated-by-the-computer?

What do the graphs tell us about the *relative strength* of the evolutionary processes the BIOL-112 students are simulating?

What tells you more about what processes occuring here: comparing the student numbers to a single critical value (11.70), or comparing the student numbers to the simulated numbers?

Imagine you added the possibility for a toothpick to mutate into a different type. What would that do to the χ^2 values? (hint: think about what part of this process causes such extreme distortions even without selection)

VIII. Extra Credit

Modify the `simDraws` function to include the possibility that toothpicks could mutate during reproduction (i.e., some fixed probability μ that an "offspring" would be a different color than their "parent"). Compare the calculated χ^2 values with mutation & drift to those with mutation, drift, & selection. The `simPop` function includes a mutation option with the same parameter name (μ is pronounced & coded as "mu"), so you can use that as a partial guide.