Week 3 Computer Lab Introduction: Genetic Diversity

Scene 1: Genetic Diversity

Welcome back! This week, we'll have another look at the Colombia spotted frog data. First, we'll do some basic checking of population genetic assumptions, including deviations from Hardy-Weinberg equilibrium, linkage disequilibrium, and the presence of null alleles. Then we'll aggregate the data in various ways to quantify genetic diversity.

This video focuses on some of the more technical challenges. Testing for Hardy Weinberg etc. involves a lot of hypothesis testing, and we'll review some important concepts including how to deal with multiple tests. Then we'll look at different approaches for manipulating data in R. Because we have unequal sample sizes, we will use rarefaction to compare allelic richness between ponds. Finally, we'll reflect on the implications that R comes with no warranty and may not always produce consistent results.

We will look at genetic differentiation in Week 4 and genetic distances in Week 5.

Scene 2: Basic Checks of Genetic Data

The first question is how variable our markers are. Here we see a simple count of the number alleles across all sampled individuals, for the first three markers of a hypothetical data set. Locus A has only one allele, which means it is monomorphic. Locus B does show variation but has few alleles only, actually 2, whereas locus C is highly polymorphic. Expected heterozygosity He is a good measure of polymorphism. He is the probability that if we sample two alleles randomly, the two alleles are different. This is similar to Simpson diversity among species. Obviously, locus A has zero expected heterozygosity, and we should drop if from the analysis because it does not contribute any useful information.

The next question is whether any of the loci show signs of null alleles. These are alleles that consistently fail to amplify in PCR and thus show up as missing values. This can occur quite frequently for example if we adapt microsatellites from a different species. It is a problem because it will bias our estimates of allele frequencies, and thus the results of many population genetic analysis methods. Here we have in the first column the estimated proportion of null alleles. We can already see that locus D has a high proportion of 21%. The second and third columns give the lower and upper limits of a 95% confidence interval. Check the second column to see whether the confidence interval contains zero. For locus D, it does not contain zero, which means that there is significant evidence for the presence of null alleles. We can either drop the locus, or redesign the primers and redo PCR.

The next thing to check is deviations from Hardy-Weinberg equilibrium. Now, there are MANY reasons a marker may be out of Hardy-Weinberg, and testing for HWE across all individuals does not make much sense because we don't expect random mating among populations. Here

we have a table of the p-value for each locus in each population. Let's consider any value below 0.05 as significant for now, these are shown in red. What we are looking for here is consistent patterns. For example, locus E is out of HWE in many populations, and population 2 is out of HWE for many markers. We may want to drop locus E, and look further into population 2 and exclude it from some analyses, or run analyses with and without it.

Another assumption of many methods is that markers are independent. This can be violated for instance if two loci are located nearby on the same chromosome and thus tend to be inherited together. Here we have a table of pair-wise comparisons between markers. We only need half of the matrix, the other half would simply mirror the same values. And we don't need the diagonal where each marker would be compared to itself. Hence the top left value is the p-value for the null hypothesis that locus B and locus C are independent. With a significance level of 0.05, we reject the null and conclude that the two markers are indeed linked. We should thus drop one. We saw earlier that locus C is much more variable than locus B, so I would suggest dropping locus B. The other marker pairs show now sign of linkage.

Taken together, we have already eliminated four markers in this made-up example! Also, we have looked at many p-values, which means that we have implicitly performed many statistical hypothesis tests, and we should probably account for this. But first, let's review hypothesis testing.

Scene 3: Hypothesis testing

Parametric Tests: From your intro stats course, you may remember different tests like the t-test of chi-square test. They all follow a similar pattern. First, we define the statistical hypothesis pair. The biological hypothesis, where we test for a certain effect like a difference or an association, should be translated into the alternative hypothesis. The null hypothesis then represents the absence of that effect, no difference or no association. Then we calculate the observed test statistic from the data. For a t-test, this is a t-statistic, for a chi-square test, a chi-square statistic, and so on. Then we compare the observed test statistic to its known distribution if the null hypothesis is true. If the value is very unusual given that distribution, we reject the null hypothesis and conclude that there is an effect. Obviously, this method is only reliable if the theoretical distribution is applicable, which means we need to check a number of assumptions and conditions for each test.

P-value < alpha? So how do we determine if the observed test statistic is unusual? We calculate its p-value and compare it to the significance level alpha, which we usually set to 5%. Here we see a density distribution that shows what values of the test statistic we should expect with what frequency if the null hypothesis is true. The area under the curve sums to 1 and represents probability. The vertical line at 1.65 is the observed test statistic. If we perform a one-sided hypothesis test with alternative 'greater', then the p-value is the red area in the right tail, that is, to the right of the line. It represents the proportion of values that are expected to be as large or larger than the observed test statistic. We can use R to calculate the p-value. Here for example the function 'pt' calculates a probability for the t-distribution. It takes as

arguments the degrees of freedom, df, and if we want the upper tail, we need to set 'lower.tail' to FALSE.

If we do a two-sided test, then we need to mirror the line on both sides and take both tails, which means that the p-value is now twice as large. In R, we would need to double the p-value that we calculated for the one-sided test. However, if we did a one-sided hypothesis test with the alternative 'less', then the p-value would be everything to the left of the observed test statistic! The lower tail is the default for the function 'pt'. As we've seen here, it really matters what type of alternative we use. It is not OK to look at the data and shop for the smallest p-value. The choice must reflect the biological hypothesis, and we need a convincing biological reason for choosing a one-sided alternative.

Permutation Tests: What if the conditions for a test are not met? R will still give you a p-value, however, this p-value is essentially garbage if the theoretical distribution does not reflect the distribution of the test statistic under the null hypothesis. If we have reason to doubt this, we could either use a non-parametric test or a permutation test. Permutation tests are very promising because they can allow us to build non-random features into the null model. They are very flexible, too: we start with the same hypothesis pair, but then we can design our own test statistic and calculate it from the data. Then comes the crucial part: we permute the data many times in a way that mimics the patterns that we expect if the null hypothesis is true. Obviously, we need to make sure that the permutation really represents the null, and this point may be hard to get past the reviewers.

To get an approximate p-value, we combine the observed value with the test statistics calculated from the permuted data, and count the proportion of values that are as extreme or more extreme than the observed value. Again, we can do a one-sided or a two-sided test.

Statistical Power: Whenever we do a hypothesis test, we may end up being right or wrong, and I'm afraid we'll never really know. Let's pretend we know the truth in this example, so we enter the table from the top. Let's assume that there is no effect in the underlying population, so we are in the left column. We performed a hypothesis test with the data from the sample, and we may have found a p-value larger than alpha and thus decided to retain the null hypothesis of no effect, which means we were right. Or we might have found a p-value that was smaller than alpha and thus rejected the null. That's the right decision given the p-value and the logic of hypothesis testing, but the wrong decision given the truth, which normally we don't know. This is a Type I error, or a false positive, and we control the rate of false positives with the significance level alpha. With an alpha of 0.05, we should expect a statistically significant result in 5% of tests, or one in 20 tests, where in truth there was no effect in the underlying population. The probabilities in each column sum to one, therefore the probability of a true negative is 1 minus alpha.

What happens when there actually is an effect in the underlying population? Now we move to the right column, and the significance level alpha is not of much help anymore. Here, we are right if we have a p-value smaller than alpha and thus reject the null and accept the alternative

hypothesis. A Type 2 error or a false negative is when based on the sample, we get a p-value larger than alpha and retain the null, although in truth there is an effect in the underlying population. We call the Type 2 error rate beta, but unlike alpha, we can't control it, and we don't really know how large it is! The probabilities sum to 1 in each column, so if the Type 2 error rate is beta, then the power of the test is 1 minus beta. The really important thing is that the probabilities here sum for each column, but not for each row. Hence the power of a test is not a function of alpha.

Indeed, the power of a test depends mostly on two things: sample size, and the size of the effect, that is, how different the group means are or how strongly associated two variables are. Here we have a figure with power curves for a two-sample t-test. On the x-axis, we have sample size, and on the y-axis, we have the power of the test. We can already see that all the blue and red lines increase with sample size. The blue lines are for one-sided tests, and the red lines for two-sided tests. Let's look at the solid lines for a large effect. In general, the blue line is higher, which means that one-sided tests are more powerful than two-sided tests. That makes sense because their p-value is twice as large. There are two horizontal lines, one at 0.8 and one at 0.95. When planning a study, we typically aim at having at least 80% power, 95% would obviously be better still. The blue line crosses the 80% line at 20, which means that we would need a sample size of 20 in each group to have 80% power to detect an effect with a one-sided t-test if the effect was large. If the effect size was medium, we use the dashed line, which means we would need 45 in each group, and for a small effect, that would be the dotted blue line and that would be way over 100 per group. So, if the underlying effect is large, a relatively small sample will do, but if the effect is small, we need a huge sample size to reliably detect it. We will learn more about what is a small, medium or large effect in next week's lab.

Now let's compare these power curves to the type I error rate alpha. Alpha does not change with sample size, which is good if we think of small samples, but it also means it is still 5% even if we have a very large sample.

Goodness of Fit Tests: Why bother about alpha and beta? This becomes especially relevant when we perform a goodness of fit test. That's the situation where we actually want to show that there is no effect. That means that the biological hypothesis now becomes the null hypothesis, and the alternative is that the data don't fit the expectation. Think of Hardy-Weinberg, linkage disequilibrium, and the presence of null alleles. In all these cases, we really perform the test to verify that there is no deviation from the expectation. In this situation, sample size is really important an in a somewhat surprising way.

What exactly is the problem? Here we see the support for the biological hypothesis as a function of sample size. In a hypothesis test, the biological hypothesis becomes the alternative hypothesis. In that case, a larger sample generally means more support for the biological hypothesis because of the increase in statistical power. In a goodness-of-fit test, however, the biological hypothesis becomes the null hypothesis, and the exact opposite happens. With larger sample size, we get less support for the biological hypothesis! With a very large sample, even a

tiny effect can be reliably detected. However, such minor deviations from Hardy-Weinberg, or a small degree of linkage, may not really be a problem.

Accounting for Multiple Tests:

Another thing to consider is the number of tests we performed and how this affects the experiment- or family-wise Type I error rate. This means, if we perform let's say 30 tests, and in truth, there is no effect for any of them, what is the chance that at least one of the p-values will be smaller than our alpha of 0.05 and thus a false positive? Here we have a plot of this cumulative Type I error rate, on the y axis, against the number of tests performed, k. Wow, if we do 100 tests, on the very right here, we have pretty much a 100% chance of getting at least one false positive. If we are mostly concerned about the false positives, we want to bring this rate back to 5%. There are a number of ways to do this, and some are implemented in the R function 'p.adjust'. The simplest is a Bonferroni correction, which corrects the p-values by multiplying them with the number of tests, k. This method is useful because you can easily calculate it for example from the information in a published paper. When you have the data, a similar but preferred method is Holm's correction, and many pop gen papers use false discovery rate. They all serve the same purpose of accounting for multiple tests to control for the overall Type I error rate.

As a side effect, these corrections reduce the statistical power of each test. So if we apply any of these corrections to our goodness of fit tests for HWE or linkage etc., we will end up with fewer significant deviations from expectations. This means that we don't need to drop as many markers. In my personal opinion, this may alleviate the symptom but does not address the real issue, which is that for goodness of fit tests, effect size is much more important than statistical significance. I think that what we need here as a field is a consensus on how much deviation is acceptable and how much is too much, and then we could test whether a deviation is larger than acceptable.

This concludes the review of hypothesis testing. A second video for this week looks at data manipulation and how to quantify genetic diversity.

VIDEO 2

Scene 1: Genetic Diversity

In this second video for Week 3, we'll look at different approaches for manipulating data in R, we'll consider unequal sample sizes and the need for rarefaction, and discuss how to deal with the fact that R functions come without warranty and may not produce consistent results.

Scene 4: Aggregating Genetic Data

Let's return to the frog data set. We have already imported it into a 'genind' object, where each row is an individual frog. We can analyze it in a number of different ways, which I'll call

‘Summarize’, ‘Filter’, ‘Group’ and ‘Aggregate’. More generally, these are data steps, or different types of data manipulation. For individuals sampled from discrete populations, like frogs sampled from different ponds, summary statistics across all loci only make sense when checking loci for linkage or null alleles. To get results for each site, we could use filtering, which means extracting all individuals from a single site, calculate the summary statistics, and maybe loop through each site. A more efficient way is grouping, where we split the dataset by population, then apply the summary function to each population. That is different from aggregating the data, where we calculate allele frequencies for each site and create a new dataset where each row is a population, not an individual.

Adegenet has different functions to support these data steps. Here I use ‘obj’ as a placeholder for the ‘genind’ object Frogs.genind. The summary function prints a convenient summary and returns it as a list, but this is for the pooled sample, not by population. Adegenet supports slicing with square brackets. Here we have three examples where we extract the first three individuals from the data set, or all individuals from population 1, or only locus A. The extracted data are returned as a new ‘genind’ object. The most important method here is ‘seppop’, which takes a ‘genind’ object and splits it into a list of ‘genind’ objects, one for each population. For aggregating allele frequencies, there is a dedicated function ‘genind2genpop’. It returns a new object type, ‘genpop’, where each row is a population, and this object type can again be used by other packages.

For most analyses, we’ll want to analyze the data by population, and the standard way is to use ‘seppop’ to split the data, then apply some function to each population, and combine the results. Adegenet does not do this for you but the authors recommend using the function ‘lapply’ from base R.

Scene 5: Your New Best Friend: ‘lapply’

‘lapply’ may seem awkward at first but has the potential to become your new best friend, so give it, or him or her, a chance!

We use ‘lapply’ to **apply** a function to each element of a **list**, so you can think of it as ‘list apply’. Obviously, it needs two arguments, the first is the list, the second is the function that we want to apply. If the list elements and the function are pretty simple, the code is easy, just provide the list and the name of the function, for example: lapply(my.list, nrow). This would calculate the number of rows, and thus the number of individuals in each population. However, in our case each list element is a ‘genind’ object, and this simple code won’t work.

The more general version looks like this: we start with lapply, and inside the bracket we indicate the list. Then we use ‘function(ls)’ to indicate that we are about to provide a function that should be applied to each element ‘ls’ of the list ‘my.list’, only then we provide the actual function, without a comma. The function must again reference ‘ls’. We could use any other name for ‘ls’, as long as we use the same name in both parts of the call. So if you want to use ‘x’ instead of ‘ls’, it would have to be ‘function(x) my.function(x)’.

The function ‘lapply’ takes a list as input and returns the results again as a list. There are a number of related functions that differ in the input and output object types. ‘sapply’ does the same as ‘lapply’ but returns a vector or matrix. This is often more convenient, but can only be used if the function returns a vector or a single value for each population. ‘mapply’ can take two or more lists. It applies the function to the first elements of all lists, then to the second elements of all lists, etc. ‘apply’ is another useful function that can be used to calculate row or column statistics for a matrix or array.

Adegenet has a function ‘propTyped’ that calculates the proportion of non-missing values. With the argument ‘by = “loc”, we specify that we want this to be calculated separately for each locus. To calculate the proportion of non-missing values for each locus in each population, we first create an object ‘tmp’ that contains a list of genind objects, one for each population. Then we can use lapply to apply the function propTyped to each population, including the argument ‘by = “loc”’. This returns the results as a list. However, for each population, the function will return a vector of 8 values, the proportion of non-missing values for each of the 8 loci. This can be combined into a matrix of loci by population hence we can use sapply as an alternative. And because ‘propTyped’ knows what to do with a ‘genind’ object, we could even use the short form.

Scene 6: R Grammar: Data Manipulation with ‘dplyr’

‘lapply’ is the proven workhorse of R data manipulation, and it is very cooperative as long as you speak its language. If that’s not your thing, there is an alternative language of data manipulation in R that is closer to the way we speak, and follows a grammar that is meant to be intuitive to learn. This is in the R package ‘dplyr’, which stands for ‘data plyers’. The functions for the data steps are actually ‘verbs’: summarise, filter, group_by, and select, to name just a few. They do pretty much what we’ve discussed before, only we haven’t seen ‘select’. It is similar to ‘filter’, but ‘filter’ extracts rows, whereas ‘select’ extracts columns.

Here are few examples of pseudo-code. The function ‘summarize_all’ takes a data frame, ‘df’, and applies each function listed inside ‘funs’ to each column in the dataframe. Here, the only function is n(), which is a function from the ‘data plyer’ package that calculates sample size. We can add more functions, and we can specify a name for the output columns generated by each function. Here sample size will be called n, and there’s another output variable being created, called valid. To understand what it calculates, we need to have a closer look at logical operators in R.

We have already seen the less than or greater than signs, we can specify less or equal, greater or equal, or with a double equation symbol, exactly equal. The exclamation mark means ‘not’, so ‘not equal’ becomes an exclamation mark followed by an equation symbol. With ‘is.na()’, we can test each value whether it is a missing value. And for the opposite, we use again the exclamation mark: ‘is not na’ test whether a value is NOT missing. Then there are other

operators for logical and, logical or, to test whether a logical value is 'TRUE', or to check whether a value is element of a given set of values.

Now we can make better sense of this function. It tests whether any value in the data frame is not missing, and takes the sum, which means it counts how many non-missing values there are. In this case, we need to add a period inside the brackets to specify that the function should be applied to each column in the data frame, not any specific column only. The remaining examples are more straight-forward: filter here extracts all individuals from population "Egg", group_by splits the dataset by population, similar to 'seppop', and select is used here to extract the columns with the loci.

There's one more twist to this: we can combine the data steps with the pipe symbol, which is two percent symbols and a right arrow in the middle. This stands for 'then do this'. Here we tell R to take the dataframe df, then to this: group the data by population, then do this: select columns A through H, which are the loci, then do this: summarize each locus by calculating the proportion of non-missing values. If you want to learn more about data manipulation with the 'data plyer' package, please check the cheat sheet, you can access it with the Add-ins in RStudio.

Scene 7: Unequal Sample Size?

With these tools, it is easy to calculate various summaries and compare them between populations. A simple question is: which population has the highest genetic diversity? There is one caveat though, and that is we may have unequal sample size, either because fewer frogs have been sampled at some ponds than at other ponds, or because some ponds have more missing values than others. Does this matter?

Let's go back to basic statistical principles. When we use statistics for hypothesis testing or estimation, we want to conclude from the sample to the underlying population. Sample size will determine how precise our estimate is. The smaller the sample size, the more variability, and thus the more uncertainty about the true value in the population from which we sampled. That in itself is not a big problem, but it affects the power of or analysis. For example, with a larger sample, we can estimate allele frequencies in each population more precisely, which makes it easier to detect genetic differentiation among populations. A bigger problem is if we introduce any bias, that means, anything that makes our estimate be off target. That's the case with allelic richness, because with more individuals, we are likely to observe more alleles, so that ponds with fewer individuals sampled are at a disadvantage from the start.

To level the playing field, we can use rarefaction. Here we see the number of alleles found, on the y axis, plotted against the number of individuals sampled, for three populations in red, green and blue. The blue population had more alleles in total, but also more individuals than the red population. Rarefaction resamples the same number of individuals from each population and thus compares the number of alleles found in each population, given a fixed sample size determined by the smallest group. This is averaged over many rounds of

resampling. You could program this yourself in R, but I recommend using the function 'allel.rich' from the 'PopGenReport' package that takes a 'genind' object, groups it by population and returns rarefied allelic richness.

Scene 8: There are several packages that build on 'genind' objects, and this makes it relatively easy to perform many analyses in R that previously would have required transferring your data between several standalone software packages. This is great, and please say thank you to the authors who have contributed functions that you use by citing the packages in your manuscripts.

In general, I prefer using pre-existing code as I may make many mistakes when I try to implement a method myself. However, R functions and packages come without any kind of warranty. This means, for example, that different implementations of the same analysis may produce different results. This has always been an issue, for example the big old-time stats packages SAS and SPSS would not necessarily produce identical results. Functions in the R 'base' packages, anything that is distributed automatically when installing R, may be considered safe. Anything in contributed packages should be considered with some caution, though.

As an example, here is Fst for the Frog dataset calculated with two different implementations in the same package, hierfstat. The two values differ by quite a bit. Most likely, this is not because of errors but because each function implements a different variant of Fst and thus uses a different formula, or sometimes it might have to do with how missing values are treated. Similarly, the analysis of molecular variance, AMOVA, has been implemented differently in different packages (ade4, pegas, and vegan), and the results may differ quite a bit.

What can you do? First, read the manual. Second, google the function and read up on it on relevant R user forums. Third, display and examine the source code. For a simple function this is straight forward: type the name of the function, without the brackets. This won't work if the function has different variants to accommodate different types of input objects. In that case, what to do depends once more on whether we are dealing with S3 or S4 objects and functions. For functions that take S3 objects, type 'methods' and the name of the method. This returns a list of the variants. Select the relevant one and type its name without brackets. For functions that take S4 objects, type 'showMethods' and the name of the method. This returns a list of the object types that the method takes. Select the relevant one. Type 'getMethod', and in the bracket, first list the name of the function, then the name of the object type, both in quotes.

Scene 1: Genetic Diversity

This brings us to the end of this long video. Who knew that genetic diversity could be so complicated? Now it is your turn in the tutorial to practice with the R functions we've covered. Then you should be well prepared to understand the worked example and adapt it to your own data set.