**Computer Lab 10 – Population structure and Bayesian assignment tests**

**Conservation Genetics (BIOL 4174 / 5174)**

Installation

All softwares in use today are freely available. We’ll install some using conda:

conda install -c bioconda structure

conda install -c conda-forge parallel

Part I: STRUCTURE

To run structure, we need to prep several parameter files as inputs. The first, (Lab10/mainparams) contains the main parameter settings for running STRUCTURE. Open this file in TextWrangler- there are several changes we’ll need to make to it:

1. Set #define BURNIN to: 20000. This is the burn-in amount for your MCMC
2. Set #define NUMREPS to: 50000. This is the post burn-in iterations for MCMC. Note that both of these are much lower than you would perform in a robust analysis
3. Set #define NUMINDS to: 211
4. Set #define NUMLOCI to: 15
5. Make sure #define PLOIDY is set to: 2
6. Make sure #define MISSING is set to: -9. This tells Structure how we’ve coded missing alleles
7. Set #define ONEROWPERIND to 1, which tells Structure the format of our input
8. Make sure #define LABEL, #define POPDATA, and #define POPFLAG are all set to 1
9. Finally, make sure #define LABEL is set to 1

We’ll set the rest of the relevant commands as arguments in our command-line call. Be sure to save your changes to the file.

Next, we need to set some things in the extraparams file, which contains aspects of the model and the prior specifications. Open in TextWrangler. The first thing we will set, is the ancestry model. There is no equivalent to jModelTest for STRUCTURE, so we’ll have to rely on prior knowledge of the study species (Bluehead Sucker) in selecting appropriate models.

Four choices are available for the ancestry model:

* NOADMIX=1: **Use No Admixture Model** – Under this model, each individual is fully assigned to one of several populations. Misapplication of this model will lead to strange results, so it should be used only when it is known that your data is comprised of non-interbreeding groups that have been separated for many generations.
* NOADMIX=0: **Use Admixture Model** – This model allows individuals to have mixed ancestry. In other words, one individual may inherit a fraction of its genome from population A, and another fraction from population B. This is a more general model that can be applied in many situations to elucidate gene flow among populations or cryptic population structure. This is also the most frequently used model and is suggested as a starting point by the manual for most analyses.
* LINKAGE=1: **Use Linkage Model** – This is an extension of the Admixture model that is designed to compensate for low levels of linkage disequilibrium. However, it is still unable to deal with markers that are very tightly linked. Using this option requires chromosomal map positions, which we don’t have- so leave it set to zero.
* USEPOPINFO=1: **Use Population Information to test for migrants** – This model can be applied when population structure reflects sampling locations with the exception of a few individuals that appear to be misclassified (i.e., potential recent immigrants into a population).

**Before setting, answer question 1 in the homework. Enter your values for** NOADMIX, **and leave** LINKAGE=0 **and** USEPOPINFO=0.

Next, we’ll set the Allele Frequency model. The settings I want you to focus on are (leave the rest as defaults):

* FREQSCORR=1: **Allele Frequencies Correlated** – this model assumes allele frequencies to be similar for different populations, possibly as a result of gene flow or shared ancestry.
* FREQSCORR=0: **Allele Frequencies Independent** – this model assumes allele frequencies are very different among populations (i.e., no gene flow and deep historical population separation).
* INFERLAMBDA=1: **Infer Lambda** – Lambda is the model parameter that specifies the distribution from which allele frequencies are sampled. Lambda is also fixed at a set value or inferred from the correlated or independent allele frequency models (this is hidden under the “Advanced” option for both models). The default option is to fix lambda = 1 for either allele frequency model. The manual recommends leaving this option alone for most analyses.

**Before setting, answer question 2 in the homework. Enter your choice for** FREQSCORR**, and leave** INFERLAMBDA=1.

Finally, you need to change 1 more thing towards the end of your extraparams file:

1. Set #define RANDOMIZE to: 0

Save both of your params files and close TextWranger.

**Running STRUCTURE with GNU-Parallel**

We will be using a shell tool called GNU-Parallel to speed up our runs by allowing us to do multiple replicates at the same time. We will be running 4 replicates each of *K* values 1 through 6. For each value of *K*, run the following command all in one line (substituting *$K* for the value):

parallel ‘structure -K $k -D 123$k45{} -m mainparams -e extraparams -i structure\_data.str -o k$k\_{}.out’ :::: <(seq 1 4)

You will need to **do this for each value of *K***. For example, the full command for *K*=1 would be:

parallel ‘structure -K 1 -D 123145{} -m mainparams -e extraparams -i structure\_data.str -o k1\_{}.out’ :::: <(seq 1 4)

A lot of things are happening in this command. Let’s unpack them:

* parallel: This portion indicates the start of a GNU-Parallel command-line call. Everything that follows in the single-quotes will be executed by GNU-Parallel. **Don’t copy and paste. Word converts these to back-ticks, which won’t work.**
* structure: This call the command-line version of Structure. What follows are arguments to be interpreted by Structure.
* -K 1: This tells Structure to run with only inferring 1 cluster (*K*)
* -D 123145{}: The -D flag is used to give Structure a ‘seed’ to start its random number generator. If different runs have the same starting ‘seed’, they will produce the same results. By inserting the K-value (in the place of $k), and the replicate number (in the place of {}- GNU-Parallel will do this for you), we ensure that all runs will have their own random number seed
* -m, -e, -i: These give Structure the paths to your mainparams file, extraparams file, and input file
* -o k1\_{}.out: Every time GNU-Parallel sees the “{}” symbol, it will insert the replicate number. We are using this functionality to have Structure name our output files (provided via -o) differently for each replicate (e.g. replicate #4 for *K* value 2 will be named k2\_4.out).
* :::: <(seq 1 4): This part tells GNU-Parallel how many times we would like it to perform the command written between single-quotes. In this case, we asked it to run the structure command 4 times: once for every number in the sequence 1-4 (e.g. 1, 2, 3, 4). If this were a “real” analysis, I recommend at least 10 replicates.

Now, go ahead and run the command for *K*=1. After the run (which should take ~30 seconds), check that files k1\_1.out\_f, k1\_2.out\_f, k1\_3.out\_f, and k1\_4.out\_f exist. Take a look at one of them using **cat** or **less** to make sure they look OK. If all goes well, do the same for the remaining *K* values {2,3,4,5,6}. Note that Structure runs for exponentially longer as you increase *K*. At the number of MCMC iterations we are sampling, I expect the *K*=6 run (the longest one) to run for only about 2-3 minutes. Luckily, we have GNU-Parallel, or else you would have to wait for each replicate to run sequentially! ☺

**Pro Tip™©:**

You can also use the following Bash **for loop** to run each *K* value sequentially, with replicates in parallel. If you are actually going to use analyses like this, I can help you get set up with AHPCC to run on the high-performance computing cluster on campus. Note that these are **back-ticks** (on ~ key) around the “seq 1 6” and **single-quotes** (on “” key) around the structure run.

for i in `seq 1 6`; do parallel ‘structure -K ${i} -D 123${i}45{} -m mainparams -e extraparams -i structure\_data.str -o k${i}\_{}.out’ :::: <(seq 1 4); done

Part II: Interpreting Structure Results

**CLUMPAK**

First, we’re going to try this the easy way. We are going to try to submit our results for parsing on the CLUMPAK server- which handles all of the downstream steps of parsing STRUCTURE results for you in one convenient and slightly broken pipeline. In my experience, the server has been incredibly finicky.

Before running CLUMPAK, we’ll need to package up our STRUCTURE results. In the directory containing your output “\_f” files, run the following to make a zip archive:

zip results.zip \*\_f

Next, navigate to the CLUMPAK web server at: <http://clumpak.tau.ac.il/>

Under **Run Main Pipeline**, click “Choose File”… Navigate to your ‘results.zip’ and upload it. Make sure STRUCTURE is checked as the format for the uploaded file (CLUMPAK can also handle ADMIXTURE-formatted results), enter your address, and click “Submit Form”.

You can track progress under **Log Data:**. Wait for ~5 minutes, if your results don’t process, call me over and hopefully we can troubleshoot.

**Interpreting CLUMPAK results**

In the results, you will receive a “job\_###\_pipeline\_summary.pdf”, as well as bar plots under **Output Images**. Next to each *K* value, CLUMPAK will report the mean log-likelihood (Mean(LnProb)), and a similarity score among replicates. *K*=1 should have similarity = 1.000 because all replicates are identical. Other *K* values likely do not. When there is much variability among replicates (as there may be in the very short runs we did today), replicates may be split into multiple bar charts- the ‘minor modes’, or less common result, will be reported lower down (scroll down).

NOTE: You may see “stripes” in the posterior probability of assignments for a population (e.g. all individuals exactly 50% assignment to Orange, 20% to Pink, 30% to Green)- this generally does not reflect any biological signal, but it a failure of the program to cluster your replicates. This can either indicate poor convergence among replicates, or that the MCMC was too short.

**Answer questions 3-6.**

Part III – Structure Harvester

Structure itself has no built-in method for determining the appropriate *K*, although its manual makes suggestions that include calculating the posterior probability of each *K*, and plotting the log likelihood of the data against each *K*. Other researchers developed the delta *K* method (Evanno et al. 2005), which has been implemented in the program Structure Harvester. This program is easily used through a web interface at <http://taylor0.biology.ucla.edu/structureHarvester/>. The original Python code (Python is pre-installed on Mac OS X) can also be downloaded from this site and run locally on your own machine. Running the program locally via command line is sometimes necessary for very large files.

**Using Structure Harvester**

Structure Harvester will perform all of the calculations described in Evanno et al. 2005, prepare graphs and summary tables, and prepare files for other programs designed to process Structure results (Clumpp and Distruct).

Go to the website <http://taylor0.biology.ucla.edu/structureHarvester/> (or Google structure harvester, and it should be the first hit), click “Browse” and upload the zipped results folder that you just created in Part II. Click “Harvest” and the website will begin analyzing the data and generating the graphs.

The first graph [L(*K*)] can be used to determine *K* via one of the methods described by the Structure manual. Examine the graph. *K* values are plotted on the X axis, while the mean ln(likelihood) of the data is on the Y axis. The point at which the ln(likelihood) reaches an asymptote is supposed to represent the number of populations present.

The fourth graph (Delta*K*) is used by the Evanno method. The *K* value that corresponds to the highest value of Delta*K* is supposed to represent the true number of populations.

**Answer the last questions in the homework.**