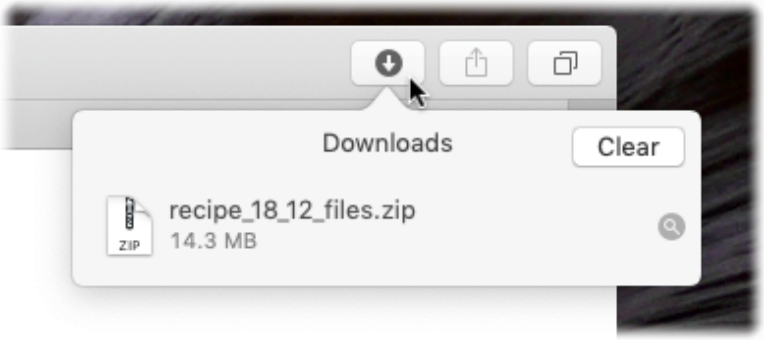
SLiM Workshop Series #22: Nucleotide I/O

1. Start by getting the VCF and FASTA files needed

Go to the Safari web browser and enter the following URL:

```
http://benhaller.com/slim/recipe_18_12_files.zip
```

Press return and the file should be downloaded. If it downloaded to your Desktop, you're good to go. If not, you need to find it and move it to the desktop, which you can do by clicking the Downloads button near the top right of the Safari window:



Click the small magnifying-glass button to the right of the pop-up, which will make the downloaded file visible in the Finder wherever Safari put it. Drag the file to the Desktop, where we will work with it.

Now double-click the `.zip` file (if it is still a `.zip` file; depending on Safari preferences it may have already been expanded to a folder). In the end, you should have a folder named **recipe_18_12_files** on your Desktop. If you double-click it to open it, you will see that it contains two files in it:

```
chr22_filtered.recode.vcf
```

```
hs37d5_chr22_patched.fa
```

The VCF file, `chr22_filtered.recode.vcf`, contains SNP data for human chromosome 22 genome sequences from the 1000 Genomes Project, from `http://www.internationalgenome.org/data`. The original download was 11.2 GB in size, which is quite large and takes a while to download, so for our purposes here it has been filtered down to sequences from Finland, retaining 99 out of 2054 sequenced individuals. SNPs that were fixed or absent among the Finland individuals were dropped, and sites representing indels were removed since SLiM doesn't support them (with a bit more work they could have been transformed into SNPs, if representing them in SLiM were important); this filtering retained 155518 out of 1103547 segregating sites. The final file is 67.9 MB – still fairly large, but manageable.

Let's take a look at its contents. On macOS, *don't* double-click it; macOS thinks that the `.vcf` filename extension is for address book contact cards, so it will do very much the wrong thing. Instead, right-click or control-click the file `chr22_filtered.recode.vcf` in the Finder, and choose **Open With** > **TextEdit.app** from the context menu. (Linux users, use `gedit` or whatever plain-text editor you prefer.) The file should open in the editor, where you can look it over; it's a pretty typical VCF file with a bunch of header lines at the top and then a large number of call lines specifying the nucleotides present in each of the 99 sequenced individuals. Each individual is diploid, so each individual is called, typically, as `0|0`, `1|0`, `0|1`, or `1|1`, depending upon whether the SNP in question is present in neither, one, or both sequenced genomes.

The URL for the reference sequence, providing the ancestral nucleotide state for every base position along the chromosome, is provided in the `##reference` header of the VCF file. The file provided here, `hs37d5_chr22_patched.fa`, is a FASTA format file derived from that original reference sequence. The sequence for chromosome 22 was extracted, reducing the file size from 3.19 GB to 52.2 MB. Sites marked as `N` in the original FASTA sequence (indicating an unknown nucleotide at that position) were changed to `A`; this shouldn't matter since SNPs are not located at such sites anyway. You can open this file in TextEdit or `gedit` to look at it; it is a typical FASTA file.

This processing of the VCF and FASTA files was done with free packages called `VCFtools` and `samtools`. You can read more details about exactly how the processing was done in recipe 18.12 in the SLiM manual, which this exercise is based upon.

2. Simulate neutral drift with empirical SNPs

To start out, we want to initialize a nucleotide-based model with the ancestral FASTA sequence. Close all open SLiMgui windows, and then make a new window and delete all of its contents.

The first thing we should do is tell SLiMgui to use the **recipe_18_12_files** folder as its working directory. Click the folder button above the output pane, at the right-hand edge:

When you click that button, a sheet will open asking you to choose the working directory. Select the **recipe_18_12_files** folder (an easy way is to drag it from the Finder onto the sheet), and then click the **Open** button.

Start the model by creating this `initialize()` callback. Don't forget about code completion with the escape key; to type the first line, for example, `iSO` and then escape gets you the `initializeSLiMOptions()` function call, and then, inside the parenthesis of the call, `n` and then escape gets you `nucleotideBased=`. Press return to accept each completion. The code:

```
initialize() {
   initializeSLiMOptions(nucleotideBased=T);
   length = initializeAncestralNucleotides("hs37d5_chr22_patched.fa");
   defineConstant("L", length);
   initializeMutationTypeNuc("m1", 0.5, "f", 0.0);
   initializeGenomicElementType("g1", m1, 1.0, mmJukesCantor(0.0));
   initializeGenomicElement(g1, 0, L-1);
   initializeRecombinationRate(1e-8);
}
```

The filename for the FASTA file can be passed directly to `initializeAncestralNucleotides()`. A path to it isn't needed, since it is in the chosen working directory.

Next, we want to create a population with an initial state loaded from the VCF file:

```
1 late() {
   sim.addSubpop("p1", 99);
   p1.genomes.readFromVCF("chr22_filtered.recode.vcf", m1);
   p1.setSubpopulationSize(1000);
}
```

This creates a new subpopulation `p1` of size `99`, which is the number of individuals contained by the VCF file. On the vector of genomes for those individuals, we then call `readFromVCF()`, providing the path to the VCF file (in the working directory, again, so no path needed). We pass it `m1` to tell it to create mutations of type `m1` from the SNPs in the VCF file. It is possible for the VCF file to explicitly specify the mutation types to be used individually for each SNP, using a SLiM-specific VCF tag, but here we just want `m1` mutations. Finally, we request a size of `1000` with

`setSubpopulationSize()`; this design essentially means that we are taking the VCF SNP frequencies as representative population-wide frequencies and expanding up to a population size of `1000` with random mating from the `99` sampled genomes. (A different initialization procedure might be better, depending upon exactly what you want to do, but this is an easy method.)

Finally, we want to generate a little output after `1000` generations:

```
1000 late() {
   catn(sim.substitutions.size() + " substitutions.");
   catn(paste(sim.substitutions.nucleotide));
}
```

The model undergoes neutral drift for a thousand generations, and then prints out the number of substitutions and then a list of the nucleotides that were substituted. Since this model uses a mutation rate of zero, those will all be SNPs present in the initial population.

If you recycle and then step, you will notice that the initialization step takes a significant amount of time; that lag is due to reading in the 52 MB FASTA file. Step again, and you will see that generation 1 also takes a substantial amount of time; that is due to reading in the 68 MB VCF file and creating all of the 155518 mutations described by it. If you now play the model, generations will proceed somewhat more quickly. It might look at first like `1000` generations will take a long time, but the model runs faster and faster as it goes because of the loss of genetic diversity due to drift; many low-frequency SNPs are being lost, and many high-frequency SNPs are fixing. Soon the model is ticking right along, and then produces its final output; for one test run, I got 8487 substitutions at the end.

3. Output a VCF file at the end

Rather than just dumping nucleotides for the substitutions to the output stream, you might wish to output a new VCF file that reflects the final state of the model. That file could then be read and analyzed by any of the various VCF-processing tools that are widely available for genomics work. To do this, change the final output event to this new code:

```
1000 late() {
   p1.genomes.outputVCF("final.vcf");
}
```

That's all that is needed. The vector of genomes in `p1` is output to the specified file (in the working directory). When running a model at the command line, by the way, the default working directory is the current directory in the shell that you run the `slim` process inside; so here, if you executed the command `cd ~/Desktop/recipe_18_12_files` in Terminal and then ran `slim` with the script file there, that would be the working directory. You can also change the current working directory in Eidos with the `setwd()` function.

So, if you recycle and run this version of the model you should see a VCF file named `final.vcf` appear in the folder at the end of the run. It's probably fairly large – 240 MB, for one test run – but you can open it in TextEdit or `gedit` if you want to see the SLiM-generated VCF output.

4. Generate a CSV file summarizing the substitutions

That VCF output provides information about the mutations still segregating at the end of the run, but it would also be nice to have a summary of the substitutions that occurred. We could use the `outputFixedMutations()` method for that, as we have seen previously, but instead let's generate a CSV file for them. (CSV stands for "comma-separated values"; it's a standard file format that just provides data as plain text separated by commas.)

The `writeFile()` function can be used to write a file, so we just need to assemble a vector of output lines. Add the following code to the end of the `1000 late()` output event:

```
subs = sim.substitutions;
lines = "position, nucleotide";
for (sub in subs)
   lines = c(lines, sub.position + ", " + sub.nucleotide);
writeFile("subs.csv", lines);
```

If you recycle and run, you will get a `subs.csv` file, which you can look at in TextEdit or `gedit`. With 7000–8000 substitutions, the performance of this code is not bad; it takes only a second or two to generate the `lines` vector and write out the file. It doesn't scale well, though, because each additional line gets added to the `lines` vector with the statement:

```
lines = c(lines, ...);
```

If there are 7500 lines already generated, adding line 7501 to the vector requires that `c()` creates a new vector of length 7500, copies all 7500 existing lines into the new vector, and then adds the new line at the end – and the same, again, for line 7502, 7503, and so forth. In computer science jargon, this is an $O(N^2)$ algorithm, which means it will get exceedingly slow as N becomes large. Those familiar with R or Python will feel a healthy urge to vectorize this code somehow. Don't worry if this is difficult to understand at first glance – this is a pretty advanced thing we're about to do – but a vectorized version of this output code would look like this instead:

```
subs = sim.substitutions;
lines = sapply(subs,
   "applyValue.position + ', ' + applyValue.nucleotide;");
lines = c("position, nucleotide", lines);
writeFile("subs.csv", lines);
```

The `sapply()` function is something like a `for` loop; it executes code once for each element in its first parameter, `subs`. The executed code is passed to it as a string; here it is the string:

```
"applyValue.position + ', ' + applyValue.nucleotide;"
```

This string is parsed by Eidos and executed by `sapply()` for each substitution in subs. When executing for a given substitution, the variable `applyValue` is defined to be that focal substitution; `applyValue` is sort of like the index variable of an Eidos `for` loop. This is also rather like the callback mechanism of SLiM; in that sense, `applyValue` is a pseudo-parameter passed to the "callback" code specified by the string. The "callback" code simply generates a string value, just as the previous code did with `sub.position + ", " + sub.nucleotide`. The `sapply()` function concatenates all of the values returned, in order, to make one big vector of lines. Then we just need to prepend the header line with `c()` and write the lines with `writeFile()`.

5. Write out a final FASTA sequence

As nucleotide-based mutations fix, they get converted to Substitution objects, and we've just written those out. The new fixed nucleotide also gets patched into the ancestral sequence; the new fixed nucleotide becomes "ancestral" since it is possessed by every individual. We might therefore want to write out the final ancestral sequence as a FASTA file; add these lines to the output event:

```
seq = sim.chromosome.ancestralNucleotides();
writeFile("anc.fa", c(">final_ancestral_seq", seq));
```

The FASTA header is added to the beginning with `c()`; no newline character is needed since `writeFile()` separates lines with newlines anyway. You can inspect `anc.fa` in TextEdit or `gedit`. Note that the sequence is one long line; you could introduce newlines into it with a bit more work.

6. BEEP! With extra time, look at section 25.2.4, detailing nucleotide-based VCF output.