Supplementary Material 1-A: Simulate SNV sequence data for pedigree founders

Nirodha Epasinghege Dona, Jinko Graham

2021-12-22

Contents

This is the first in a series of RMarkdown documents describing how we simulated exome-sequencing data in pedigrees ascertained to have four or more relatives affected with lymphoid cancer. The overall workflow for this project is shown below.

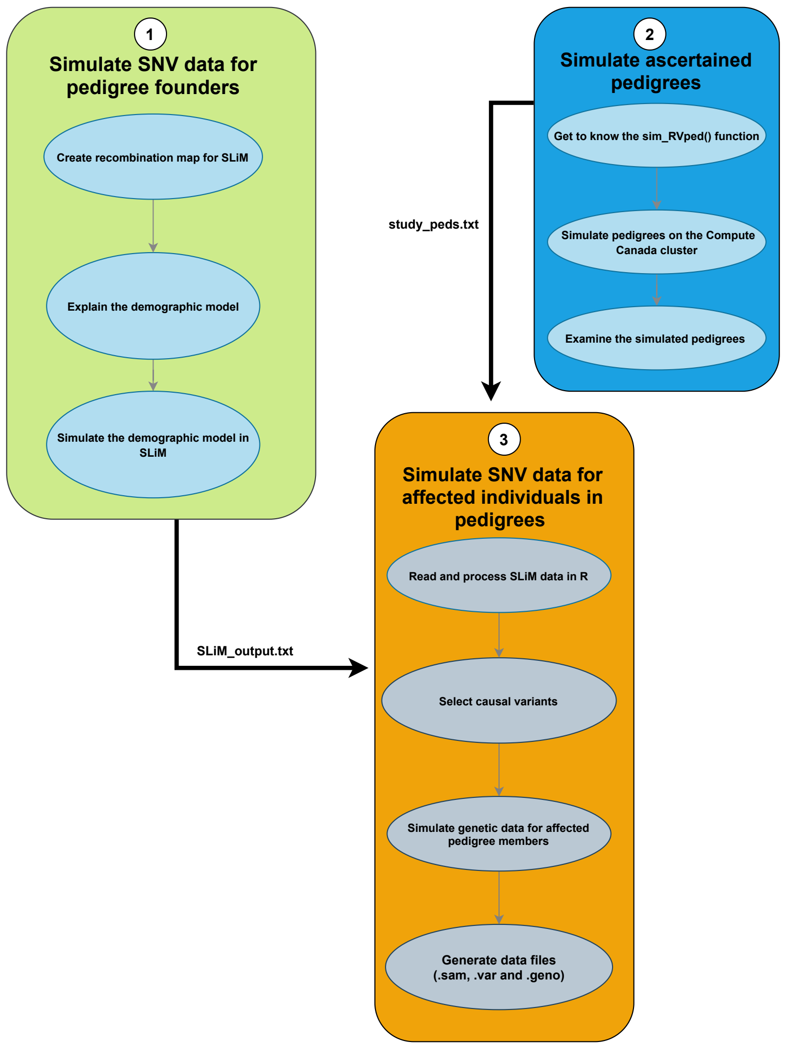

1
Simulate SNV data for pedigree founders
Create recombination map for SLiM
Explain the demographic model
Simulate the demographic model in SLiM
2
Simulate ascertained pedigrees
Get to know the sim_RVped() function
Simulate pedigrees on the Compute Canada cluster
Examine the simulated pedigrees
study_peds.txt
SLiM_output.txt
3
Simulate SNV data for affected individuals in pedigrees
Read and process SLiM data in R
Select causal variants
Simulate genetic data for affected pedigree members
Generate data files (.sam, .var and .geno)

Figure 1: Work-flow for simulating the exome-sequencing data for ascertained pedigrees.

This document focuses on the part of the flowchart labelled as 1 (the green box). To start, we require single-nucleotide variant (SNV) sequences for pedigree founders. These founders are assumed to be sampled from an American Admixed population, which we simulate with the evolutionary simulation package `SLiM` (Haller et al. 2019). In particular, we simulate genome-wide sequences of exons only, to mimic exome sequencing.

The outline of this document as follows. Section 1 explains how we create the `SLiM` recombination map using the `create_SlimMap()` function in the `SimRVSequence` R package (Nieuwoudt, Brooks-Wilson, and Graham

2020). Section 2 explains the demographic model for the source population of the pedigree founders. Section 3 discusses how we set the parameters in our SLiM model to simulate the exon-only SNV sequences.

The final outcome of this RMarkdown document is the file `SLiM_output` containing SNV exon sequences from a simulated American Admixed population. In the third RMarkdown document of this series, the population sequences file is sampled to get the founder sequences to drop down the ascertained pedigrees. The final gene-dropping step generates the exome-sequencing data in the family-based study of lymphoid cancer families.

1 Create recombination map for SLiM

To simulate the genome-wide exon-only sequences with SLiM, we need to supply a recombination map which reads the exon positions in chromosomes. We use the `create_SlimMap()` function in the `SimRVSequence` (Nieuwoudt, Brooks-Wilson, and Graham 2020) R package, as shown in the next code chunk.

```
library(SimRVSequences)

# Load hg_exons data set in SimRVSequence package
data("hg_exons")

# Create recombination map for exon-only data using the hg_exons dataset
s_map <- create_slimMap(exon_df = hg_exons)
head(s_map)
```

```
##   chrom segLength  recRate mutRate  exon simDist endPos
## 1     1     11873 0.00e+00   0e+00 FALSE       1      1
## 2     1       354 1.00e-08   1e-08  TRUE     354    355
## 3     1       385 3.85e-06   0e+00 FALSE       1    356
## 4     1       109 1.00e-08   1e-08  TRUE     109    465
## 5     1       499 4.99e-06   0e+00 FALSE       1    466
## 6     1      1609 1.00e-08   1e-08  TRUE    1609   2075
```

We use the `hg_exons` dataset in the `SimRVSequence` package to specify the exon positions of each of the 22 human autosomes, based on the hg38 reference genome from the UCSC Genome Browser (Nieuwoudt, Brooks-Wilson, and Graham 2020). As shown above, the call to `create_SlimMap()` returns a data frame with information about the genetic segments in each chromosome. As an example, the first row in the output above represents information about the genetic segment before the first exon on chromosome 1. The second row represents information about the first exon on chromosome 1. The exon contains 354 base pairs and the recombination and mutation rates in this exon are 10^{-8} per site per generation. The other columns of the data frame are described in the `SimRVSequences` documentation. The recombination rate between adjacent exons is set to the number of base pairs in the intervening intronic segment (`segLength`) multiplied by 10^{-8} per base pair per generation (`recomb_rate`). Further, the gap between two unlinked chromosomes is set to be a single base pair and the recombination rate between them is set to be 0.5 per base pair per generation (Harris and Nielsen 2016). Since we are interested in exon-only data, the mutation rate outside exons is set to zero and mutation rates inside exons is set to 10^{-8} per base pair per generation (Nieuwoudt, Brooks-Wilson, and Graham 2020).

We need three variables from `s_map` to create the recombination map for simulating exon-only data by SLiM: `recRate`, `mutRate` and `endPos`. We select these three variables and shift the `endPos` variable forward by one unit because SLiM reads arrays starting at position as 0 rather than 1. We save the resulting output as a text file (`Slim_Map_chr.txt`) to be used as a recombination map for SLiM.

```
# Restrict output to the variables required by SLiM
slimMap <- s_map[, c("recRate", "mutRate", "endPos")]

# Shift endPos up by one unit
```

```r
slimMap$endPos <- slimMap$endPos - 1

# Print first four rows of slimMap
head(slimMap, n = 4)
```

```
##    recRate mutRate endPos
## 1 0.00e+00   0e+00      0
## 2 1.00e-08   1e-08    354
## 3 3.85e-06   0e+00    355
## 4 1.00e-08   1e-08    464
```

```r
# Write the results to a text file
write.table(slimMap, file ="Slim_Map_chr.txt")
```

The next section explains the demographic model we will use to simulate the population-level, exon-only SNV sequences. These sequences will be randomly sampled from the population to be assigned to the founders of our ascertained pedigrees in later steps of the workflow.

2 Explain the demographic model

Demographic models play a major role in understanding the genetic patterns in human populations. Throughout human evolution, different demographic events such as expansion, migration, splitting etc. have occurred, affecting genetic diversity (Ragsdale and Gravel 2019). The population-genetics literature has several established demographic models inferred from genetic data (Gutenkunst et al. 2009). Some of these models have been compiled in `stdpopsim`, a standard library of population-genetic simulation models (Adrion et al. 2020). At the time of writing, this library contains around nine demographic models. Among these, we select the **American Admixture** demographic model of Browning et al. (2018) because the family-based study motivating our work is in a North American population.

2.1 American admixture demographic model

In the American-Admixture model (Browning et al. 2018), the pre-admixture model parameters are selected from the Out-of-Africa model of Gravel et al. (2011) illustrated below.

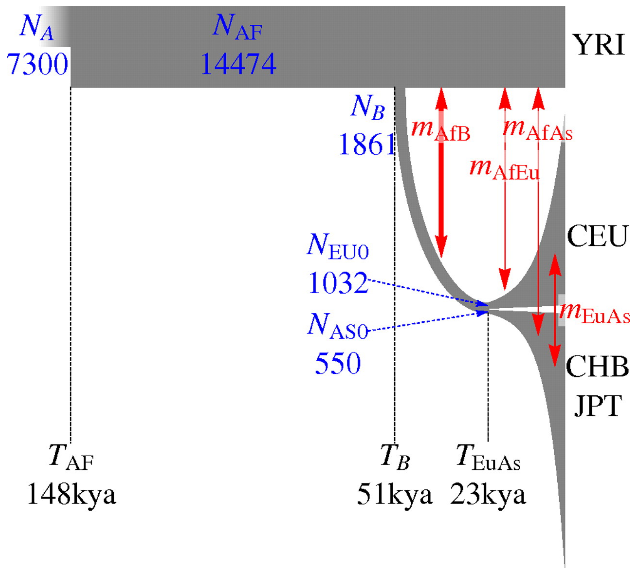


Figure 2: The inferred Out-of-Africa demographic model.

In the figure, the parameter estimates have been rounded and times are expressed in kilo-years before present (kya). The demographic model has three populations representing Africa, Europe and Asia. The initial effective population size of Africa was 7310 individuals which then increased to $14,474$ individuals 5920 generations ago (148 kya, assuming a generation time of 25 years). About 2040 generations ago (51 kya), the out-of-Africa migration event occurred with a migrating effective population size of 1861 individuals. Then migration occurred between Africa and out-of-Africa populations with a rate of 1.5×10^{-4} per generation. About 920 generations ago (23 kya), the out-of-Africa population split into two populations, Europe and Asia, with effective sizes of 1032 and 554 individuals, respectively. These two populations then grew at rates of 3.8×10^{-3} per generation for Europe and 4.8×10^{-3} per generation for Asia. Further, between these three populations, (Africa , Europe and Asia) migrations occurred. The migration rates per generation were 2.5×10^{-5} between Africa and Europe, 7.8×10^{-6} between Africa and Asia, and 3.11×10^{-5} between Europe and Asia (Browning et al. 2018). Admixing started about 12 generations ago (0.3 kya) with the initial effective size of the admixed population being $30,000$ individuals. The growth rate of the admixed population was 5% per generation with $\frac{1}{6}$ of the admixed population originating from African ancestry, $\frac{1}{3}$ from European ancestry and $\frac{1}{2}$ from Asian ancestry (Browning et al. 2018).

As described in the next section, we use SLiM together with the inferred American-admixture demographic model to simulate population-level exon-only SNV sequences.

3 Simulate the demographic model in SLiM

The following SLiM script generates genome-wide exon-only SNV sequences for a population under the American-Admixture demographic model. The script is a `.slim` file, which we have embedded in an R code chunk (that is not run). The original `SLiM_American_Admixture.slim` file can be found on our GitHub page at https://github.com/SFUStatgen/SeqFamStudy/.

```
initialize() {

// Seed number which helps to reproduce the same result
setSeed(2181144364021);

// Read recombination map created by SimRVSequence R package

lines = readFile("~/Slim_Map_chr.txt");
Rrates = NULL;
Mrates = NULL;
ends = NULL;

for (line in lines)
{
components = strsplit(line);
ends = c(ends, asInteger(components[3]));
Rrates = c(Rrates, asFloat(components[1]));
Mrates = c(Mrates, asFloat(components[2]));
}
Exomelength = ends[size(ends)-1];

initializeRecombinationRate(Rrates, ends);

initializeMutationRate(Mrates, ends);

initializeSex("A"); // Specifies modeling of an autosome

initializeMutationType("m1", 0.5, "g", -0.043, 0.23); //non-synonymous
initializeMutationType("m2", 0.5, "f", 0.0); // synonymous

m1.mutationStackPolicy = "l";
m2.mutationStackPolicy = "l";

initializeGenomicElementType("g1", m1, 1); // positions 1 and 2
initializeGenomicElementType("g2", m2, 1); // positions 3

starts = repEach(seqLen(asInteger(round(Exomelength/3))) * 3, 2) +
  rep(c(0,2), asInteger(round(Exomelength/3)));
end_pos = starts + rep(c(1,0), asInteger(round(Exomelength/3)));
types = rep(c(g1,g2), asInteger(round(length(starts)/2)));

initializeGenomicElement(types, starts, end_pos);

}

// Initialize the ancestral African population
1 { sim.addSubpop("p1", asInteger(round(7310.370867595234))); }

// End the burn-in period; expand the African population
73105 { p1.setSubpopulationSize(asInteger(round(14474.54608753566))); }

// Split Eurasians (p2) from Africans (p1) and set up migration
76968 {
sim.addSubpopSplit("p2", asInteger(round(1861.288190027689)), p1);
p1.setMigrationRates(c(p2), c(15.24422112e-5));
p2.setMigrationRates(c(p1), c(15.24422112e-5));
}

// Split p2 into European (p2) and East Asian (p3); resize; migration
78084 {
sim.addSubpopSplit("p3", asInteger(round(553.8181989)), p2);
p2.setSubpopulationSize(asInteger(round(1032.1046957333444)));
p1.setMigrationRates(c(p2, p3), c(2.54332678e-5, 0.7770583877e-5));
p2.setMigrationRates(c(p1, p3), c(2.54332678e-5, 3.115817913e-5));
p3.setMigrationRates(c(p1, p2), c(0.7770583877e-5, 3.115817913e-5));
}

// Set up exponential growth in Europe (p2) and East Asia (p3)
78084:79012{
t = sim.generation - 78084;
p2_size = round(1032.1046957333444 * (1 + 0.003784324268)^t);
p3_size = round(553.8181989 * (1 + 0.004780219543)^t);
p2.setSubpopulationSize(asInteger(p2_size));
p3.setSubpopulationSize(asInteger(p3_size));
}

// Create the admixed population
79012 early(){
sim.addSubpop("p4", 30000); //This new subpopulation is created with 30000 new empty individuals
p4.setMigrationRates(c(p1, p2, p3), c(0.1666667, 0.3333333, 0.5));
}
```

```
//After this early() event, SLiM will generate offspring, and the empty individuals in p4 will be
// discarded and replaced by migrant offspring from p1, p2 and p3 as requested.
79012 late(){
p4.setMigrationRates(c(p1, p2, p3), c(0, 0, 0));
}

// Set up exponential growth in admixture (p4)
79012:79024 {
t = sim.generation - 79012;
p4_new_size = round(30000 * (1 + 0.05)^t);
p4.setSubpopulationSize(asInteger(p4_new_size));
}

// Output and terminate
79024 late() {
p4.individuals.genomes.output(filePath = "~/SLiM_output.txt");
}
```

Before the simulation starts, we need to initialize the mutation rate, recombination rate, genomic structure and so forth as the simulation parameters (Haller et al. 2019). We read the recombination map into SLiM using the `readFile()` function. Inside this function, we supply the path to our recombination map text file. Then we create three null vectors named `Rrates`, `Mrates` and `ends` to save the recombination rates, mutation rates and end positions of each exon in our recombination map, respectively.

Next, we use a `for`-loop to move along the genome, reading each line of the recombination map and:

- save the recombination rate, mutation rate and end position of each genomic segment,
- initialize the recombination rate for each genomic segment with the `initializeRecombinationRate()` function, by specifying the rate and the end position of the genomic segment,
- initialize the mutation rate for each genomic segment with the `initializeMutationRate()` function,
- specify that the genomic segment belongs to an autosomal chromosome with the `initializeSex()` function,
- specify the mutation type for each genomic segment with the `initializeMutationType()` function (see below),
- specify the mutation stacking policy for each genomic segment with the `mutationStackPolicy` command (see below),
- specify the type for each genomic segment with the `initializeGenomicElementType()` function (see below).

In exons, the last base-pair position in a three base-pair codon (coding for an amino acid in a protein) is a synonymous site. Synonymous sites are viewed as selectively neutral in comparison to the first two base-pair positions in a codon, which are non-synonymous. Therefore, we simulate two types of mutations: synonymous and non-synonymous. The `initializeMutationType`("m1", 0.5, "g", -0.043, 0.23) callback in the for-loop explains all the parameters that are held by the "m1" mutation type. We use "m1" to represent the non-synonymous mutations. These non-synonymous mutations have a dominance coefficient of 0.5 and the selection coefficient is generated from a gamma distribution with mean -0.043 and shape parameter is 0.23 (Harris and Nielsen 2016). We initialize the synonymous mutations separately with another call to the `initializeMutationType()` function. In `initializeMutationType`("m2", 0.5, "f", 0.0) callback, the "m2" mutation type represents the synonymous mutations and they have a fixed selection coefficient denoted by "f". The selection coefficient of this type of mutation is always 0, as seen in the fourth argument of the function. The dominance coefficient in the second argument of the function is 0.5.

In SLiM (as in biology), the individuals rather than the mutations are under selection. Selection acts on the individual, through their fitness value. The fitness value of an individual is calculated from the fitness effects of all the mutations carried by that individual (based upon their selection coefficient, dominance coefficient, and heterozygous/homozygous state). All the fitness effects are multiplied together to produce the individual fitness. The individual fitness value then affects selection. Specifically, in the default Wright-Fisher (WF) model of SLiM (which we use), lower fitness means a lower probability of mating. As a result, deleterious mutations tend to decrease in frequency and beneficial mutations tend to increase in frequency.

SLiM allows for recurrent mutations at a given base position on a given sequence (Haller et al. 2019). By default, SLiM "stacks" any mutations that occur in the same location as pre-existing mutations on a given sequence. This default behaviour of "mutation stacking" ("s" for stacked), is changed to "l" (last) with the command `m1.mutationStackPolicy = "l"`, so that new mutations occurring in the same location as pre-existing mutations on a given sequence replace the pre-existing mutations.

The next initialization task is to create the chromosome structure. In SLiM we can model different genomic structures in the chromosomes. We consider exons only, which have two genomic element types: one for non-synonymous sites (base positions 1 and 2 of a codon) and the other for synonymous sites (base position 3 of a codon). These genomic element types are called "g1" and "g2" and alternate as g1, g2, g1, g2, g1, etc. along the exome until the end position of a chromosome is reached. The first genomic-element type corresponds to non-synonymous sites, is initialized as "m1" and could have mutations with selection coefficients that come from the negative gamma distribution. The second genomic element type corresponds to the synonymous sites, is initialized as "m2" and could have neutral mutations. We use the `initializeGenomicElementType()` function to specify these two genomic elements and our exome structure. For example, `initializeGenomicElementType("g1", m1, 1)` specifies that genomic element type "g1" is defined as using mutation type "m1" for all of its mutations. The second genomic element type "g2" is defined as using mutation type "m2" for all its mutations. Then we create the alternating start and end positions of the "g1" and "g2" genomic elements along the exome. Finally, we initialize the two genomic elements "g1" and "g2" with `initializeGenomicElement()`, supplying their starting and ending positions along the exome.

After the `initialize()` callbacks end, we run our simulation under the Out-of-Africa model described in the SLiM manual (Haller et al. 2019), with exact parameter estimates from Gravel et al. (2011). In the first generation, the African ancestral population, labelled "p1", is created with the function `sim.addSubpop()` and initial effective population size of 7310 individuals. Haller et al. (2019) start the model at 79024 generations back from the present (gbp) and set it to be generation 0 in the forwards simulation. The simulation then takes 10*`African ancestral population size` generations as the neutral burn-in time (Haller et al. 2019). At generation 73105 in the simulation (5919 gbp), the ancestral population, "p1", increases in effective size from ~ 7310 to ~ 14474 individuals. Subsequently, at generation 76968 of the simulation (2056 gbp), the African ancestral population, "p1", splits into the Eurasian ancestral sub-population, "p2", and migration starts between these two subpopulations. The command `p1.setMigrationRates` sets the migration rate from the African to the Eurasian ancestral sub-population, while `p2.setMigrationRates` sets the migration rate from the Eurasian ancestral to the African sub-population. Then at 78084 generations in the simulation (940 gbp), the "p2" Eurasian sub-population splits into European and Asian sub-populations. We create a new sub-population,"p3", to represent the Asian sub-population and let the Eurasian ancestral sub-population become the European sub-population. After the Eurasian ancestral population becomes the Asian and European sub-populations, we allow for migration between the African, Asian and European sub-populations, setting the migration rates according to the literature. Then, starting from 78084 generations in the simulation (940 gbp), we specify exponential growth in European (p2) and Asian (p3) sub-populations, until 79012 generations in the simulation (12 gbp). At 79012 generations (12 gbp), we create the American admixed sub-population with an initial effective population size of 30000 individuals and set the migration rates between the admixed and the other three sub-populations according to Browning et al. (2018). Once the admixed sub-population is created, migration into and out of it is stopped and it grows exponentially at rate 5% per generation until the present at 79024 generations into the simulation. Finally in generation 79024 of the simulation (the present) we terminate our SLiM simulation and collect the output.

We only consider the SLiM output for the American admixed sub-population. We extract the genomic sequences of all individuals in the admixed sub-population with the function `p4.individuals.genomes.output()` obtaining output formatted as follows:

```
#OUT: 79024 GS 107752 /project/6007536/epasiedn/SLiM/American_Admixture/SLiM_output.txt
Mutations:
7229 50171 m2 51287555 0 0.5 p1 5 60626 ...
13218 484904 m2 39812003 0 0.5 p1 45 9536 ...
5202 762125 m2 36490340 0 0.5 p1 70 64099 ...
...
Genomes:
p*:0 A 0 1 2 3 4 5 6 7 8 9 10 11 ...
p*:1 A 10605 1 2 3 10606 4 5 6 8 10607 10608 9 ...
p*:2 A 10605 1 2 4 15639 15640 6 10608 15641 15642 15643 15644
p*:3 A 0 1 2 19096 19097 4 6 19098 19099 9 10 19100
...
```

In the above output, the first line starts with "# OUT:", followed by the generation of the simulation (79024) from which the output is obtained. Then "GS" tells us the data is formatted as "genomes SLiM format" and this is followed by the number of haploid genomes (2* number of individuals). Finally, the full path where we save the output is printed.

The second line of the output starts the mutation section. In the mutation section, each line represents a mutation which is currently segregating in the population and the nine fields on a line represent the mutation properties. The first field is the SLiM-generated identifier number which helps to identify the mutation easily within the program. The second field is the mutation's identification number. The third field represents the type of the mutation. The fourth field is the base-pair position of the mutation on the chromosome. The fifth and sixth fields represent selection and dominance coefficients, respectively. The seventh field is the sub-population in which the mutation originated. The eighth field is the generation of the simulation when the mutation arose. Finally, the ninth field represents the number of copies of the mutation in the sub-population.

The last section in the output represents the genomes section. In the genome section, a line corresponds to a haploid genome in the sub-population. For example, in the first line, "p*: 0", means the 0th genome of the sub-population . Then "A" represents autosome, the type of the genome. This is followed by the SLiM-generated identification numbers of all the mutations carried by this haploid genome. Recall that the SLiM-generated identification numbers are in the first field of the mutation section.

3.1 Simulation on Compute Canada Cluster

This SLiM simulation is highly memory intensive and not suitable for most personal computers. We therefore use the Compute Canada cluster (http://www.computecanada.ca) as described next. First, we need to install the SLiM software. The way we install the software is exactly the same as
how we install the software on our own computer. Use the SLiM manual guidelines for this task. After we install SLiM, we use a job scheduler on the compute cluster to run our jobs. On the Compute Canada Cluster, the job scheduler is the **Slurm Workload Manager** . Slurm helps to allocate resources and time, and provides methods to execute our work. To run the SLiM script we write the following Slurm script, `job_serial.sh`:

```
#!/bin/bash
#SBATCH --account=def-jgraham
#SBATCH --ntasks=1
#SBATCH --time=7-05:05:00
#SBATCH --mem=64000M

module load StdEnv/2020 gcc/9.3.0 slim/3.4.0

slim SLiM_American_Admixture.slim
```

The content of `job_serial.sh` is described as follows.

- `#SBATCH-account=def-jgraham` specifies the account name. In this example, the account name is "def-jgraham".
- `#SBATCH-ntasks=1` defines the number of processors. We request 1 processor to run the program.
- `#SBATCH-time=7-05:05:00` specifies the time limit for the job. To avoid the simulation being stopped prematurely for running over the allocated time, we try to err on the side of allocating too much.
- `#SBATCH-mem=640000M` specifies memory required for our simulation. We request 64GB. Next, the executable commands are aligned in the script file:
- `module load StdEnv/2020 gcc/9.3.0 slim/3.4.0` loads the SLiM version we installed on the Cluster.
- `slim SLiM_American_Admixture.slim` calls SLiM to run the SLiM script, `SLiM_American_Admixture.slim`.

To submit the slurm script to the cluster, we type the following in our log-in node:

```
[epasiedn@gra-login2 American_Admixture]$ sbatch job_serial.sh
```

This SLiM simulation took approximately 3 days to complete on the Compute Canada cluster. The SLuRM script allocated 64GB to run the simulation. Out of this 64GB, the job utilized 43.61GB. The output of this SLiM simulation is saved as `SLiM_output.txt` and we use this file as one of the inputs for the third step of our workflow. We will return to the `SLiM_output.txt` output in our third RMarkdown document.

3.2 Summary Statistics

We use R and the output in `SLiM_output.txt` to obtain summary statistics for the American admixed population. We start by reading `SLiM_output.txt` into R.

```
library(SimRVSequences)
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ---------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(Matrix)
```

```
##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
# Read the SLiM output text file to R
# Note:Change the path for the file as necessary.
exDat <- readLines("/Users/jgraham/OneDrive - Simon Fraser University (1sfu)/NirodhaStuff/Data/SLiM_outp
```

The file size of `SLiM_output.txt` is approximately 6 GB and takes approximately 1 minute to load into R on a Windows OS with an i7-8550U @ 1.8GHz,16GB of RAM.

```
# Read the mutations and genomic sections in the output
MutHead <- which(exDat == "Mutations:")
GenHead <- which(exDat == "Genomes:")

# Get the population count in sequences
popCount <- as.numeric(unlist(strsplit(exDat[1],
                                       split = " ", fixed = TRUE))[4])

# Population count in individuals
popCount/2
```

```
## [1] 53876
```

The number of individuals in the simulated American admixed population is 53,876.

```
# Extract mutation data from SLiM's Mutation output
# only retaining the tempID, type, position,
# selection coefficient and count of each mutation
MutOut <- do.call(rbind, strsplit(exDat[(MutHead + 1):(GenHead - 1)], split = " ", fixed = TRUE))
MutData <- data.frame(tempID = as.numeric(MutOut[, 1]),
                      type = MutOut[, 3],
                      position = as.numeric(MutOut[, 4]),
                      selCoef = as.numeric(MutOut[, 5]),
                      count = as.numeric(MutOut[, 9]),
                      stringsAsFactors = TRUE)

nrow(MutData)
```

```
## [1] 862243
```

The number of mutations segregating in the American admixed population is 862,243. We next examine what percentage of these have derived allele frequency less than 1% in the population.

```
# Add 1 to temp ID so that we can easily associate mutations to columns.
# By default SLiM's first tempID is 0, not 1.
MutData$tempID <- MutData$tempID + 1
# First position in SLiM is 0, not 1
MutData$position <- MutData$position + 1

# Calculate the population derived allele frequency.
# Divide the allele count by the population size.
MutData$afreq <- MutData$count/(popCount)

# Get the percentage of SNVs whose allele frequency < 0.01
af_less <- which(MutData$afreq < 0.01)
af_less_per <- length(af_less)/ nrow(MutData)

af_less_per
```

```
## [1] 0.9426565
```

Among the 862,243 mutations segregating in the simulated American-admixed population, approximately 94% have derived allele frequencies less than 1%. This is slightly less than the approximately 97% of single-nucleotide variants observed to have alternate allele frequences less than 1% in the TopMed study of the American population (Taliun et al. 2021).

Next, we check the percentage of mutations that are singletons in the simulated American admixed population.

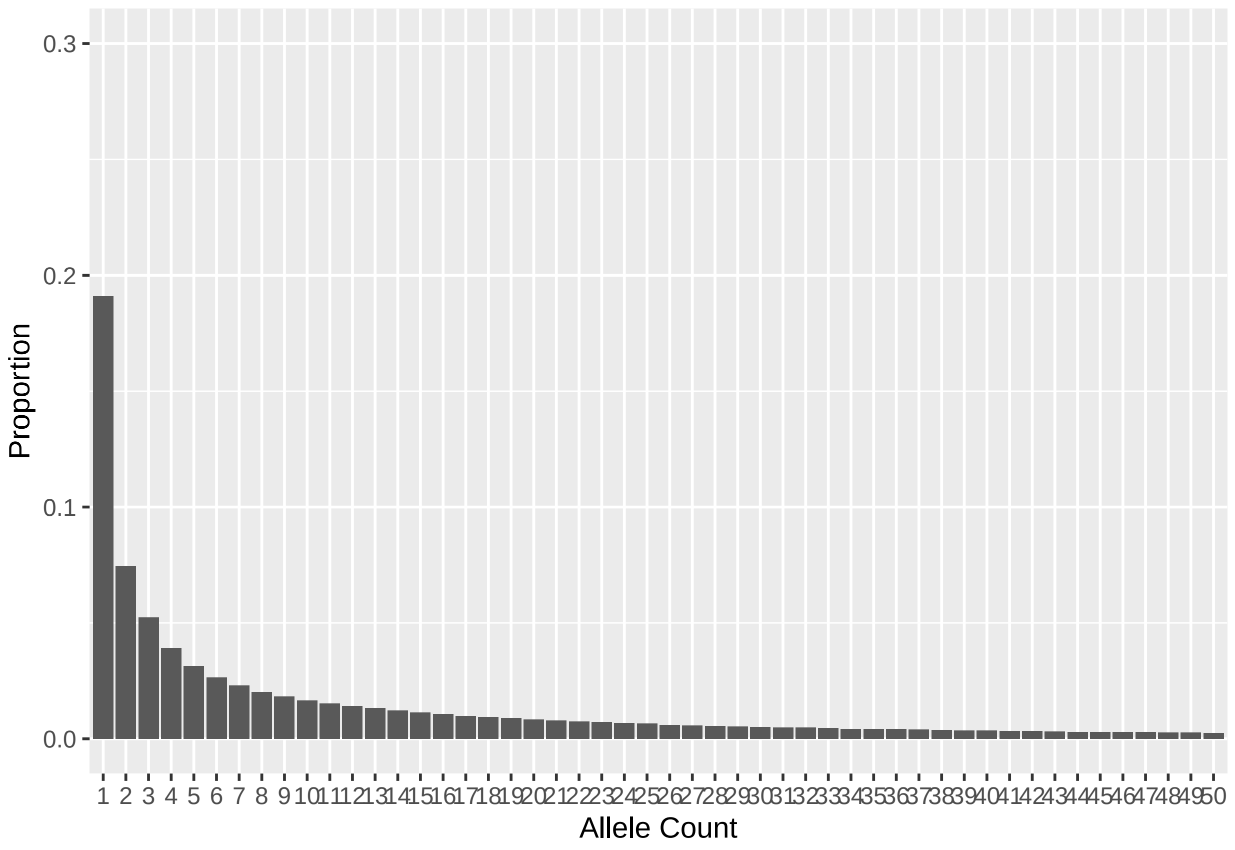
```
# Use the prevalence (the number of times that the mutation occurs in any genome)
# column in MutData dataframe to calculate the singleton percentage
singleton <- MutData %>%
  count(count) %>%
  mutate(percentage = n/nrow(MutData))

colnames(singleton) <- c("number_of_allele", "count", "proportion")
head(singleton)
```

```
##   number_of_allele  count proportion
## 1                1 164624 0.19092530
## 2                2  64401 0.07469008
## 3                3  45210 0.05243301
## 4                4  33804 0.03920473
## 5                5  27019 0.03133571
## 6                6  22881 0.02653660
```

Among the 862,243 mutations, only 19% are singletons. By contrast, in the TopMed study, about half the variants are singletons (Taliun et al. 2021). The following figure illustrates the allele frequency spectrum in the simulated population.

```
# Plot the first 50 sites in the allele frequency spectrum
ggplot(singleton) +
  geom_bar(mapping = aes(x = as.factor(number_of_allele),
                                            y = proportion),
                              stat="identity",
                              position="dodge") +
  xlab("Allele Count") +
  ylab("Proportion") +
  ylim(0, 0.3) +
  scale_x_discrete(limits= as.character(1:50))
```



We believe that our simulated American-admixed population has fewer rare variants and singletons than the TopMed population because it lacks TopMed's variety of source populations. Our SLiM simulation has only three source populations and we collect mutation data from only the American-admixed population. By contrast, the TopMed study considers the entire American population consisting of many more source populations as well as an admixed population. To investigate this hypothesis, we combined data from all four populations in our SLiM simulation to see if the singleton percentage increased. Due to the high computational cost, we simulated mutations for chromosomes 8 and 9 only. Combining all four populations, the percentage of singletons increased from 19 to 26% of mutations. An additional RMarkdown document, for supplementary material 1-B, discusses the commands to generate and summarize all the source populations and check the proportions of singletons, after combining the four populations.

References

Adrion, Jeffrey R, Christopher B Cole, Noah Dukler, Jared G Galloway, Ariella L Gladstein, Graham Gower, Christopher C Kyriazis, et al. 2020. "A Community-Maintained Standard Library of Population Genetic Models." Edited by Graham Coop, Patricia J Wittkopp, John Novembre, Arun Sethuraman, and Sara Mathieson. *eLife* 9 (June): e54967. https://doi.org/10.7554/eLife.54967.

Browning, Sharon R., Brian L. Browning, Martha L. Daviglus, Ramon A. Durazo-Arvizu, Neil Schneiderman, Robert C. Kaplan, and Cathy C. Laurie. 2018. "Ancestry-specific recent effective population size in the Americas." *PLoS Genetics*. https://doi.org/10.1371/journal.pgen.1007385.

Gravel, Simon, Brenna M. Henn, Ryan N. Gutenkunst, Amit R. Indap, Gabor T. Marth, Andrew G. Clark, Fuli Yu, Richard A. Gibbs, and Carlos D. Bustamante. 2011. "Demographic history and rare allele sharing among human populations." *Proceedings of the National Academy of Sciences of the United States of America*. https://doi.org/10.1073/pnas.1019276108.

Gutenkunst, Ryan N., Ryan D. Hernandez, Scott H. Williamson, and Carlos D. Bustamante. 2009. "Inferring the joint demographic history of multiple populations from multidimensional SNP frequency data." *PLoS Genetics*. https://doi.org/10.1371/journal.pgen.1000695.

Haller, Benjamin C, Philipp W Messer, Yoann Buoro, Deborah Charlesworth, Jeremy Van Cleve, Jean Cury, Michael Degiorgio, et al. 2019. "SLiM : An Evolutionary Simulation Framework CookBook," no. May.

Harris, Kelley, and Rasmus Nielsen. 2016. "The genetic cost of Neanderthal introgression." *Genetics*. https://doi.org/10.1534/genetics.116.186890.

Nieuwoudt, Christina, Angela Brooks-Wilson, and Jinko Graham. 2020. "SimRVSequences: An R package to simulate genetic sequence data for pedigrees." *Bioinformatics* 36 (7): 2295–97. https://doi.org/10.1093/bioinformatics/btz881.

Ragsdale, Aaron P., and Simon Gravel. 2019. "Models of archaic admixture and recent history from two-locus statistics." *PLoS Genetics* 15 (6): 1–19. https://doi.org/10.1371/journal.pgen.1008204.

Taliun, Daniel, Daniel N. Harris, Michael D. Kessler, Jedidiah Carlson, Zachary A. Szpiech, Raul Torres, Sarah A.Gagliano Taliun, et al. 2021. "Sequencing of 53,831 diverse genomes from the NHLBI TOPMed Program." *Nature* 590 (7845): 290–99. https://doi.org/10.1038/s41586-021-03205-y.