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

Nirodha Epasinghege Dona, Jinko Graham

2021-12-14

Contents

- The following figure illustrates flow chart for the whole project.

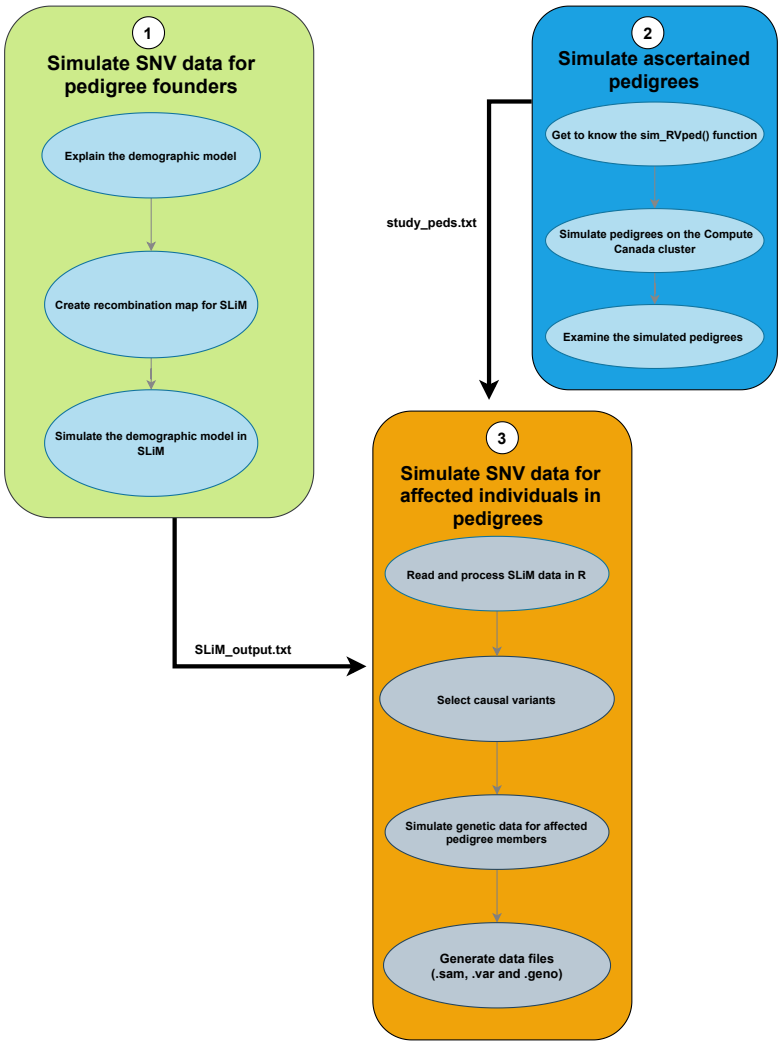


Figure 1: The flow chart which represents the entire work flow of simulating the sequence data for the pedigrees.

- We are focusing on the part which is labelled as 1 (the green box) through this R-markdown document.
- In-order to simulate sequence data for sample of pedigrees, we need to simulate single-nucleotide variant (SNV) data for pedigree founders of the sampled pedigrees.
- For this we use `SLiM` (Haller et al. 2019) which is a reliable evolutionary simulation package.
- `SLiM` allows us to simulate genome-wide sequence data forward in time for a large population of individuals.
- In our study, we consider genome-wide exon-only data to mimic an exome sequencing family study.

- This document explains all the steps that we use to simulate population SNV data for pedigree founders using `SLiM`.
- The outline of this document as follows.
- Section 1 explains the demographic model from the literature that we use to represent our population of founders in sampled pedigrees.
- Section 2 explains how we create the `SLiM` recombination map using the `create_SlimMap()` function in the `SimRVSequence` (Nieuwoudt, Brooks-Wilson, and Graham 2020) R package.
- In section 3, we discuss our SLiM model to simulate the SNV data under the demographic model.
- The Final outcome of this R-markdown document is `SLiM_output` data file which gets used by our R-markdown file 3 to generate the exome sequencing data in family base study.

1 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. occurred and these events affected the genetic diversity (Ragsdale and Gravel 2019).
- In the population-genetics literature, there are several established demographic models inferred from genetic data.
- As pointed out by Gutenkunst et al. (2009), we can use existing demographic models for the sampling designs of our studies.
- We consider the demographic models in `stdpopsim` (Adrion et al. 2020), the standard library of population genetic simulation models.
- In this library, there are around nine demographic models.
- Among these models, we select the **American Admixture** demographic model of Browning et al. (2018) because of our interest in the Northern American population.
- We next explain details of the American admixture demographic model.

1.1 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) .
- The following figure illustrates the Out-of-Africa model which is adopted from Gravel et al. (2011).

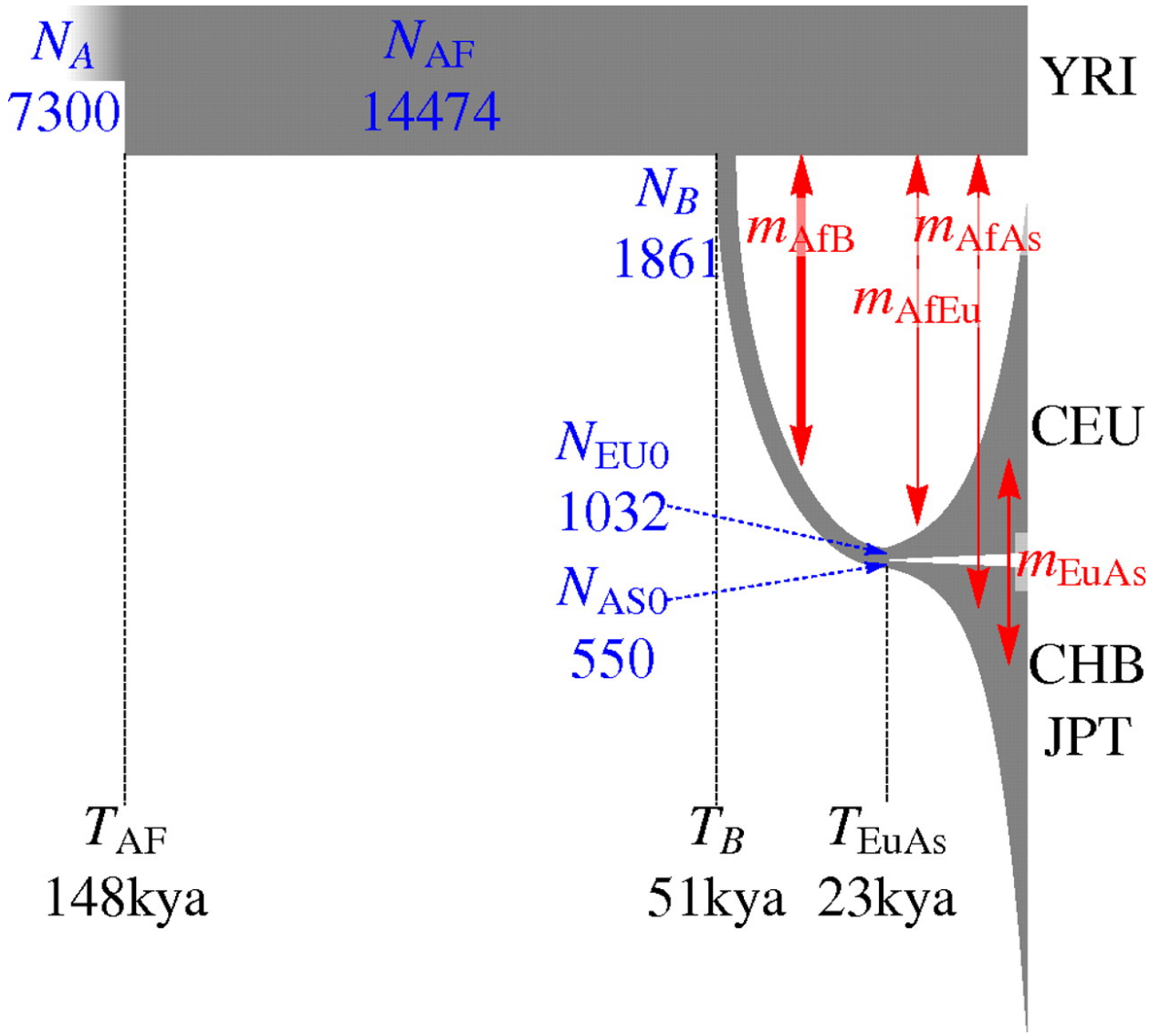


Figure 2: The representation of the inferred Out of Africa model.

- Figure 2 is a representation of the inferred Out-of-Africa model (Gravel et al. 2011). Note that the parameter estimates in the figure 2 are not the exact values but rather rounded values.
- In the Out-of-Africa model, there are three populations; Africa, Europe and Asia.
- The initial population size of Africa was 7310 individuals which then increased to $14,474$ individuals 5920 generations ago.
- 2040 generations ago the out-of Africa migration event occurred with a migrating population size of 1861 individuals.
- Then migration occurred between Africa and out-of-Africa with a rate of $1.5 * 10^{-4}$ per generation.
- 920 generations ago, the out-of-Africa population split into two populations, Europe and Asia with sizes of 1032 and 554 individuals, respectively.
- Then these two populations started to grow with the 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).
- The admixing is started 12 generations ago with the $30,000$ individuals initial size of the admixed population.
- The growth rate of the admixed population is 5% and $\frac{1}{6}$ of the population of African ancestry, $\frac{1}{3}$ of European ancestry and $\frac{1}{2}$ of Asian ancestry (Browning et al. 2018).
- We use the inferred American admixture demographic model and discuss how to set it up in SLiM in section 3.
- Our next aim is to create the recombination map which is required for the SLiM simulation. The next section explains how we create the recombination map.

2 Create recombination map for SLiM

- To simulate the genome-wide exon-only SNVs from SLiM, we need to supply the recombination map which reads the exon positions in chromosomes.
- For this, the `create_SlimMap()` function in the `SimRVSequence` (Nieuwoudt, Brooks-Wilson, and Graham 2020) R package helps us to create the required recombination map.
- The following code chunk explains how we create the required recombination map for SLiM.

```
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` data set in the `SimRVSequence` package.
- The hg38 reference genome with the UCSC Genome Browser is used to collect the data in `hg_exons` (Nieuwoudt, Brooks-Wilson, and Graham 2020).
- The `hg_exons` data set presents the positions of each of the 22 human autosomes.
- By supplying the `hg_exons` data set to the `create_SlimMap()` function, we get the recombination map required for the SLiM model.
- As shown in the above output of this function, it returns a data frame describing the 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 of this exon is 10^{-8} per site per generation. The other columns of the data frame are described in the `SimRVSequences` documentation.

- The recombination between adjacent exons is set to the number of base pairs in the segment (`segLength`) multiplied by 10^{-8} per base pair per generation (`recomb_rate`). Further, the gap between two unlinked chromosomes is defined as 1 base pair and the recombination rate between them is set to 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).
- From the output of `create_SlimMap()`, we need three variables to create the recombination map for simulating exon-only data by SLiM: `recRate`, `mutRate` and `endPos`.
- We select these three variables from the above output to supply the recombination map to SLiM. Also, it is important to shift the `endPos` variable forward 1 unit, because the `Eidos` scripting language in SLiM, reads arrays starting position as 0 instead 1.

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

# Shift endPos up by one unit
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
```

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

- As shown in the above R code chunk, we select only the three variables which are required for the recombination map in the SLiM simulation.
- We save the resulting output as a text file (`Slim_Map_chr.txt`) which we can use as our recombination map to read in SLiM.
- In the next section, we describe our SLiM model which we use to simulate exon-only SNV data for our population of unrelated pedigree founders.

3 Simulate the demographic model in SLiM

- The SLiM manual (Haller et al. 2019) explains the Out-of-Africa model of Gravel et al. (2011).
- We use this Out-of-Africa SLiM script to develop the American admixture model explained in Browning et al. (2018).
- The following SLiM script is for the American admixture model.
- Let's discuss the SLiM model in more detail.
- Note: The following code chunk is not in R but rather is a `.slim` file. To provide a clear representation of the SLiM script to the readers, we use an R code chunk to represent it. However, you can find the original `SLiM_American_Admixture.slim` file in our GitHub page.

```
initialize() {
```

```
// Seed number which helps to repoduce the same result
setSeed(2181144364021)

// Read recombination map cretated 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 admix 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 = "~/Output_Full.txt");
}
```

- In SLiM modelling, the first step is to initialize the required parameters to begin the simulation.
- The `initialize()` callback does this task.
- 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).
- Let's discuss the `initialize()` callbacks in our SLiM model.
- Our first step inside this `initialize()` callback is to load the recombination map we designed in R using the `create_SlimMap()` function.

- 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 as `Rrates`, `Mrates` and `ends` to save the recombination rates, mutation rates and end positions of each exon in our recombination map, respectively.
- We use a `for` loop to read each line of the loaded text file of our recombination map and save the three variables in the above mentioned three vectors.
- We initialize the recombination rates along the whole exome. The `initializeRecombinationRate()` function is used to specify the recombination rates and end positions of these rates. The first parameter is the recombination rates and the second parameter is the positions of the exons on the chromosomes.
- We initialize the mutation rates as well. The `initializeMutationRate()` function is used for this task.
- `initializeMutationRate()` initializes the mutation rates along the chromosomes.
- Inside the `initializeMutationRate()` function, we use the the vector of mutation rates along the chromosomes as the first parameter and the positions of the exons in the chromosomes as the second parameter.
- The `initializeSex()` function defines the type of the chromosome that we need to simulate. Since we model autosomes we define it as "A."
- The next initialization callback is the mutation type. For this, first we use the `initializeMutationType()` function to create different types of mutations.
- In exons, the last base-pair position in a 3 base-pair "codon" (coding for an amino acid in a protein) is called 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 here. They are synonymous and non-synonymous.
- The `initializeMutationType`("m1," 0.5, "g," -0.043, 0.23) callback 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).
- Then for the synonymous mutations, we again initialize them separately by using another `initializeMutationType` function.
- In `initializeMutationType`("m2," 0.5, "f," 0.0), 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 mutations is always 0, as seen in the fourth argument of the function. The dominance coefficient in the second argument is 0.5.
- In SLiM (as in biology), mutations themselves are not under selection; individuals are under selection. The selection is done based on the fitness value of the individuals.
- The fitness value of an individual is calculated from the fitness effects of all mutations carried by that individual (based upon selection coefficient, dominance coefficient, and heterozygous/homozygous state). All the fitness effects get multiplied together to produce the individual fitness.
- The individual fitness value then affects selection; in Wright-Fisher (WF) models (this is the default setting in SLiM), lower fitness means a lower probability of mating.
- Our simulation is designed under the default model, the Wright-Fisher model.

- As a result, deleterious mutations tend to decrease in frequency and beneficial mutations tend to increase in frequency.
- In SLiM recurrent mutations at a given base position can occur (Haller et al. 2019).
- In SLiM the default behavior is "mutation stacking" referred to as "s"(stacked). In our model this default behavior 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 genome 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.
- In our study, we only consider the exons in our chromosomes.
- Exons have two genomic element types, one for non-synonymous sites (positions 1 and 2 of the resulting codon) and the other for synonymous sites (position 3 of the resulting codon).
- We called these genomic element types "g1" and "g2" and they would 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 (corresponding to the non-synonymous sites) could have mutations with selection coefficients that come from the negative gamma distribution (which we initialize as "m1" to represent the non-synonymous mutations).
- The second genomic element type (corresponding to the synonymous sites) could have neutral mutations (which we initialize as "m2" to represent the synonymous mutations).
- We use the `initializeGenomicElementType()` function to specify these two genomic elements and our exome structure.
- `initializeGenomicElementType("g1", m1, 1)` specifies that genomic element type "g1" is defined as using mutation type "m1" for all of its mutations.
- We define the second genomic element type "g2" using mutation type "m2" for all its mutations.
- Then we code to create the exome structure creating start and end positions of "g1" and "g2" genomic elements.
- As the final initialization task, 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 the first generation of our simulation.
- We use the Out-of-Africa model of Gravel et al. (2011) that is described in the SLiM manual (Haller et al. 2019). Note that the SLiM manual uses the exact parameter estimates from the original model described in Gravel et al. (2011).
- In the first generation the first sub population is created with the inferred initial size and we label it as "p1."
- We use the function `sim.addSubpop()` to add "p1," the new sub population, by giving its initial size.
- Haller et al. (2019) start the model at 79024 generations back (approximately 1.98 million years); this is generation 0 in the model.
- The model then takes 10*`African ancestral population size` generations as the neutral burn-in time (Haller et al. 2019). An expansion of the African population occurs at 73105 generations.
- Therefore, at 73105 generations, we change the sub-population size of "p1" (Africa) from ~ 7310 to ~ 14474.
- After this event, at generation 76968, the African sub-population splits into the Eurasian ancestral sub-population.

- We create another sub-population, "p2," to represent the Eurasian ancestral sub-population and migration is started between these two populations (Haller et al. 2019).
- `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, the "p2" Eurasian sub-population splits into European and Asian sub-populations. Therefore, we create a new sub-population,"p3," to represent the Asian sub-population. The Eurasian ancestral sub-population becomes the European sub-population.
- After that, we set the migration rates between three sub-populations accordingly.
- Then we set up exponential growth in European (p2) and Asian (p3) sub-populations starting from 78084 generations to 79012 generations.
- At 79012 generations, we create our admixed sub-population with an initial size 30000 and set the migration rates between the admixed and the other three sub-populations as given in Browning et al. (2018). After we create the admixed sub-population, we stop all migrations.
- Then between 79012 to 79024 generations we set exponential growth of this admixed sub-population with a rate of 5% (individuals per generation).
- Finally in generation 79024 we terminate our SLiM simulation and collect our output from the model.
- Since our main study is based on the Northern American population, we only consider the American admixed sub-population as the output.
- Therefore, we use the function `p4.individuals.genomes.output()` to get the all individual's genomes of the admixed sub-population.
- The following example represents the format of the output returned from the `p4.individuals.genomes.output()` function.

#OUT: 79024 GS 107752 /project/6007536/epasiedn/SLiM/American_Admixture/Output_Full.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 example output, the first row starts with # OUT: and this is followed by the generation (79024) in which the output is generated. Then "GS" represents the "genomes SLiM format" and this is followed by the sample size in number of genomes (2* number of individuals). Finally, the full path where we save the output is printed.
- Then the second line starts the mutation section.
- In the mutation section each row represents a mutation which is currently segregating in the population and the nine columns represent the mutation properties.
- The first column is the SLiM-generated identifier number which helps to identify the mutation easily. The second column is the mutation's identification number. The third field represents the type of the mutation. The fourth column is the base-pair position of the mutation on the chromosome. Fifth and sixth columns represent selection and dominance coefficients, respectively. The seventh column

is the sub-population in which the mutation originated. The eighth column is the generation when the mutation arose. Finally, the ninth column 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 row corresponds to a genome in the sub-population and specifies all the mutations carried by the genome.
- Because we use `p4.individuals.genomes.output()`, we get all the genomes in p4 sub-population. The first line in the genomes section in the above example, "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 genome. Recall that the SLiM-generated identification numbers are in the first column in 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 in the cluster to run our jobs.
- In the Compute Canada Cluster we use the **Slurm Workload Manager** as the job scheduler.
- Slurm helps to allocate resources and time, and provides methods to execute our work.
- To run the SLiM script we use a Slurm script as follows.

```
#!/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
```

- Let's understand each line of the above batch script file.
- `#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. Usually, we allocate a time which is more than the expected time to run the program. This is because, if our simulation takes a longer time than the allocated time, the simulation will stop without executing when the given time limit is achieved.
- `#SBATCH-mem=640000M` - specifies memory that we require to run our simulation. We request 64GB.
- Next, the executable commands are aligned in the script file. They are:
- `module load StdEnv/2020 gcc/9.3.0 slim/3.4.0` - loads the SLiM version which we installed.
- `slim SLiM_American_Admixture.slim`- calls for SLiM to run the SLiM script file: `SLiM_American_Admixture.slim`. This is the final command in the slurm script file.

- Then we use `sbatch` command to submit above slurm script file to run in the cluster. We type the `sbatch` command in our log in node as follows.

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

- This simulation took approximately 3 days to complete.
- In our job script we allocate 64GB to run the simulation. Out of this 64GB, the job utilized 43.61GB.
- The output of this SLiM model is saved as `SLiM_output.txt` and we use this file as one of the inputs for the third phase of our work flow.
- We will examine the `SLiM_output.txt` output in our third R-markdown document.

3.2 Summary Statistics

- We obtain summary statistics of the American admixed population by using the SLiM output.

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

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
```

```
## -- 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
exDat <- readLines("D:/SFU_Vault/SLiM_Output/Output_Full.txt")
```

- The `SLiM_output.txt` file size is approximately 6 GB.
- We read the `SLiM_output.txt` file to R and it takes approximately 1 minute to load to 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 size of 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
```

- There are 862,243 number of mutations are currently segregating in the admixed population.
- Then we examine among these 862,243 mutations the percentage of 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 these mutations, approximately 94% have frequencies less than 1%.
- We compare these results with the TopMed study (Taliun et al. 2021). The TopMed study has newly released empirical results for the American population and suggests that the majority of variants have

alternate allele frequencies of less than 1%. According to Taliun et al. (2021), approximately 97% of variants in the TopMed Study are rare (i.e. have allele frequencies less than 1%).

- In our American admixed population also, we find that approximately 94% of the mutations are rare.
- We use the following code chuck to check the singleton percentage in the American admixed population.

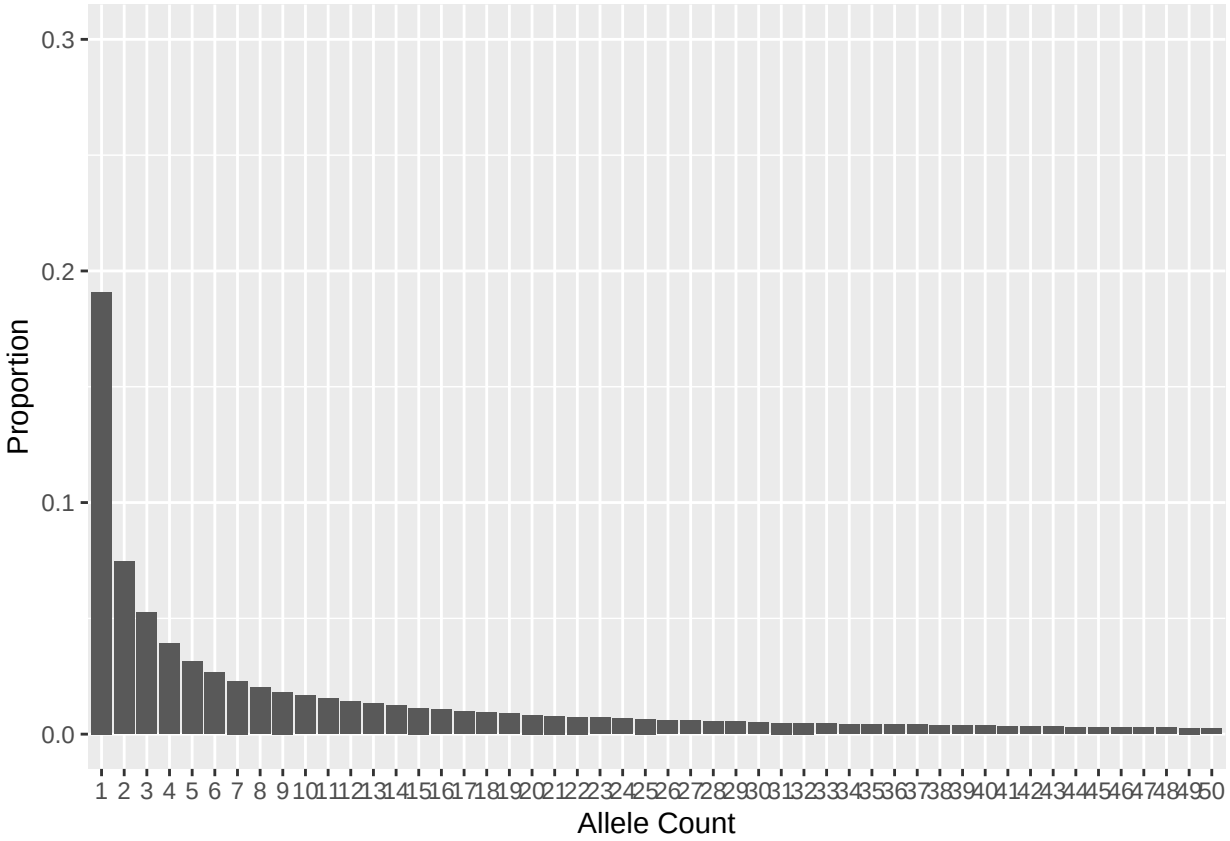
```
# use the prevalence (the number of times that the mutation occurs in any genome)
# column in MutData dataframe to calculate the singleton percentage
singelton <- MutData %>%
  count(count) %>%
  mutate(percentage = n/nrow(MutData))

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

```
##   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 862,243 mutations, 19% of them are singletons. The following figure illustrates the allele frequency spectrum.

```
# plot the first 50 sites in the allele frequency spectrum
ggplot(singelton) +
  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))
```



- In the TopMed study, about half of the variants are singletons (Taliun et al. 2021).
- The reason for the discrepancy between our results and TopMed is our lack of source populations. In our SLiM model we have only three source populations and we collect SNV data from the American admixed population only. But in the TopMed study, they consider the entire American population which consists of many more source populations.
- To check this, we combine data from all four populations in our SLiM model and check if the singleton percentage is increased. Due to the high computational cost we only simulate data from chromosome 8 and 9.
- Combining all four populations, around 26% of variants are singletons.
- The 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.

Reference

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.