Supplementary Material 1-B: Combining Source Populations with the American-Admixed Population in SLiM

Nirodha Epasinghege Dona, Jinko Graham

2021-12-21

Supplementary Material 1-A discusses the SLiM simulation of the American-admixed population and the processing of its data. This document discusses how to obtain and process the data not just for the American-admixed population but also for all four populations in the simulation. We focus on chromosomes 8 and 9 only to reduce computational cost, and alter the original SLiM script as follows: 1. at the top of the script, we provide an abbreviated recombination map, `Slim_Map.txt`, containing only those lines of the original recombination map (in the data frame, `s_map`) pertaining to chromosomes 8 and 9, and 2. at the bottom of the script, we call `sim.outputFull()` rather than `p4.individuals.genomes.output()` to obtain the output. Otherwise, the SLiM script remains exactly the same as the original in Supplementary Material 1-A.

```
initialize() {

// Read abbreviated recombination map for chromosomes 8 and 9.

lines = readFile("~/Slim_Map.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{
p2_new_size = p2.individualCount;
p3_new_size = p3.individualCount;
defineConstant("pop_size", c(p2_new_size, p3_new_size));
sim.addSubpop("p4", 30000);
p4.setMigrationRates(c(p1, p2, p3), c(0.1666667, 0.3333333, 0.5));
}
79012 late(){
p4.setMigrationRates(c(p1, p2, p3), c(0, 0, 0));
}
```

```
// Setup exponential growth in Europe (p2) and East Asia (p3)
79012:79024 {
t = sim.generation - 79012;
p2_new_size = round(pop_size[0] * (1 + 0.003784324268)^t);
p3_new_size = round(pop_size[1] * (1 + 0.004780219543)^t);
p4_new_size = round(30000 * (1 + 0.05)^t);
p2.setSubpopulationSize(asInteger(p2_new_size));
p3.setSubpopulationSize(asInteger(p3_new_size));
p4.setSubpopulationSize(asInteger(p4_new_size));
}

// Output for all populations (not just p4) and terminate
79024 late() {
sim.outputFull("~/SLiM_output_chr8&9.txt");
}
```

We read `output_all_chr8&9.txt` into R and obtain the number of individuals in each population and in all populations combined.

```
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
# Note:Change the path for the file as necessary.
exDat <- readLines("D:/SFU_Vault/SLiM_Output/SLiM_output_chr8&9.txt")
```

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

# Get the population count for each source population

popCount_1 <- as.numeric(unlist(strsplit(exDat[PopHead + 1], split = " "))[2])
popCount_2 <- as.numeric(unlist(strsplit(exDat[PopHead + 2], split = " "))[2])
popCount_3 <- as.numeric(unlist(strsplit(exDat[PopHead + 3], split = " "))[2])
popCount_4 <- as.numeric(unlist(strsplit(exDat[PopHead + 4], split = " "))[2])


# Get the total population count
popCount <- popCount_1 + popCount_2 + popCount_3 + popCount_4
```

The following table of population sizes summarizes the output of the above commands.

Table 1: Population sizes.

Population	size
African	14,475
European	35,815
Asian	48,765
Admix	53,876
Total	**152,931**

```
# Extract mutation data from SLiM's Mutation output
# only retaining the tempID, type, position, selection coefficient and prevalence of each mutation
MutOut <- do.call(rbind, strsplit(exDat[(MutHead + 1):(IndHead - 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] 142549
```

On chromosomes 8 and 9, the number of mutations segregating in all four populations is 142,549.

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

# Get the percentage of SNVs whose derived-allele frequency is < 0.01
```

```r
af_less <- which(MutData$afreq < 0.01)
af_less_per <- length(af_less)/ nrow(MutData)

af_less_per
```

```
## [1] 0.9616693
```

Among the 142,549 mutations on chromosomes 8 and 9, approximately 96% have frequencies less than 1%.

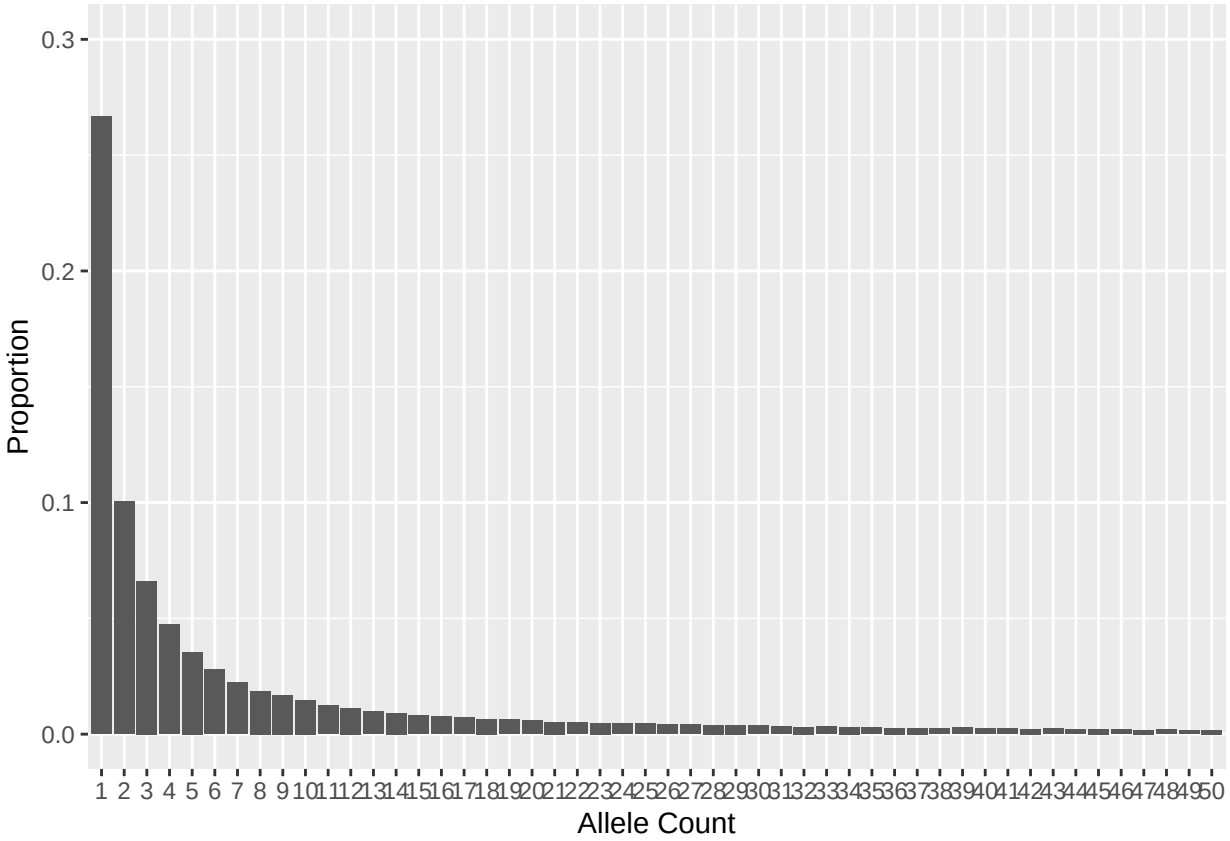
In Supplementary Material 1-A, we discuss how 26% of the variants in the combined populations were singletons. The following commands are used to calculate this percentage.

```r
# 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 38012 0.26665918
## 2                2 14312 0.10040056
## 3                3  9436 0.06619478
## 4                4  6731 0.04721885
## 5                5  5051 0.03543343
## 6                6  3979 0.02791321
```

The following figure illustrates the derived-allele frequency spectrum for chromosomes 8 and 9.

```r
# Plot derived-allele counts of up to 50 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))
```


0.3
0.2
0.1
0.0
Proportion
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
Allele Count