SLiM Notes

October, 2023

Table of Contents

# 1 Syllabus

# 2 SLiM Guide

Notes on usage of SLiM for evolutionary modeling. Adapted from the SLiM manual: <http://benhaller.com/slim/SLiM_Manual.pdf>

## 2.1 Common SLiM Functions

### 2.1.1 Initialize functions

Set a mutation rate: initializeMutationRate(numeric rates, [Ni ends = NULL], [string$ sex = "\*"])

## 2.2 Introduction

Coding concepts to introduce

1. line termination with ;
2. code blocks, brackets
3. vectors (c())
4. Data Types
5. Functions and Arguments
6. Scientific Notation
7. Function signatures (currently in initialize section)

Bio concepts?

1. Mutation types - SNPs, indels
2. Genomic regions - centromeres, telomeres, introns, exons, etc.

## 2.3 A Basic SLiM Simulation

Our first simulation will be a lightly modified version of the script that comes up when we first open the SLiM graphical user interface (GUI). It is a simple simulation tracking one population, with no selection:

// set up a simple neutral simulation  
initialize() {  
 initializeMutationRate(1e-7);  
   
 // m1 mutation type: neutral  
 initializeMutationType("m1", 0.5, "f", 0.0);  
   
 // g1 genomic element type: uses m1 for all mutations  
 initializeGenomicElementType("g1", m1, 1.0);  
   
 // uniform chromosome of length 100 kb with uniform recombination  
 initializeGenomicElement(g1, 0, 99999);  
 initializeRecombinationRate(1e-8);  
}  
  
// create a population of 500 individuals  
1 early() {  
 sim.addSubpop("p1", 500);  
}  
  
2000 late() {   
 sim.outputFixedMutations();   
}

## 2.4 Initliazing a Simulation

The first step in running a simulation is defining all of the parameters that characterize our populations - how many populations are we studying? How big are they? What do individual genomes look like? We use an initialize() block to define any parameters like this.

In almost every simulation, we will define a core set of common parameters. They are:

### 2.4.1 Global Mutation Rates

The mutation rate is set by the command initializeMutationRate(). In its simplest form, we run this function with a single argument, the mutation, as such: initializeMutationRate(1e-7);. Now, when executing the simulation, SLiM will go through each gamete base by base, and will introduce a mutation with a probability of 1e-7.

### 2.4.2 Mutation Types

When running our simulation, we can distinguish between multiple kinds of mutations, each with their own prevalence and impact - for example, a neutral substitution, or a lethal deletion, or a rare variant that confers a selective advantage. To create a mutation type, we use the initializeMutationType() command. For example, this following line of code creates a deleterious mutation:

initializeMutationType("m1", 0.5, "f", -0.02);

Let’s break the arguments to initializeMutationType down:

* the **mutation id**: this can be any integer, or a string in the format m<integer>. This is the name that you will use to keep track of your mutation.
* the **dominance coefficient**: Here, 0.5. This is used when determining how a mutation impacts the fitness of your individual. A mutation with a dominance coefficient of 1.0 is completely dominant; a mutation with a dominance coefficient of 0.0 is recessive. Mutations with values in between represent incomplete dominance - in our case, 0.5 indicates that heterozygotes have half the fitness effect of the mutation
* the last two arguments, "f" and -0.02 denote the **distribution** and **magnitude** of **how our mutation impacts fitness**. f indicates that the effect is *fixed* - that is, the mutation impacts all affected individuals identically. The value -0.02 indicates that this effect is a fitness reduction of 0.02 - any individual homozygous for this allele is expected to have offspring at 0.98 the frequency of an individual without this allele, all else being equal. While this allele has the same effect on all carriers, we can also create alleles with different fitness effect distributions - for example "n" indicates that for each individual with this mutation, we draw the fitness effect from a normal distribution, "e" indicates that we draw fitness effects from the exponential distribution. For some of these, e.g. the normal distribution, we may need to provide more than one numerical parameter.

### 2.4.3 Types of DNA

Now that we have defined the types of mutations we can encounter in our simulation, we can no specify the types of genomic regions that we are working with. For example, we can create genomic regions that represent telomeres, centromeres, introns, exons, etc. And intuitively, we can anticipate that mutations are likely to have different effects in different regions (for example, a SNP in an exon is more likely to have a dramatic phenotypic impact than a SNP in a centromere).

We can define a type of DNA using the initializeGenomicElementType() function. This function takes three argument, in the following order:

* the **name** of the genomic region (in the form "g" + an integer)
* the **kinds of mutations** that can occur in this type of DNA region.
* the **relative frequencies** of differnet mutation types

Here is a simple example of this command:

initializeGenomicElementType("g1", m1, 1.0);

This creates a type of DNA region called "g1", which can only have m1 type mutations. Since there is only one type of mutation, it happens at a frequency of 1.

### 2.4.4 Genomic Regions

Now that we have definied a type of DNA that we can see in our genome, we need to great a specific instance of it. This is done with the initializeGenomicElement() command. This takes as arguments:

* the *type of genomic element*
* the *start coordinate*
* the *stop coordinate*

For example:

initializeGenomicElement(g1, 0, 99999);

This creates a genomic element of type g1, which extends from bases 0 to 99999.

#### 2.4.4.1 Recombination Rates

Lastly, we need to set the recombination rate. This is done using the initializeRecombinationRate() function, which functions similarly to how we defined mutation rates. For this function, we just provide a float defining the per base recombination rate:

initializeRecombinationRate(1e-8);

## 2.5 Running a simulation

Now that we have initialized a simulation, we can define the events that happen during our simulation. Our simulation is organized into generations (used interchangeably with tick cycles), and within each generation, a predefined set of events happen in a fixed order:

1. If you have defined any events as occurring 'early' in a generation, they are executed first
2. Offspring are generated
3. Fixed mutations are removed in the offspring - that is, if all individuals have two copies of a mutation, SLiM stops keeping track of it
4. Parents die, offspring become the new parents
5. If you have defined any events as occurring 'late' in a generation, they are now executed

### 2.5.1 Adding subpopulations

We have used the initialize block to precisely define how the DNA of all of our organisms is structured. Now, we can create a subpopulation. This is conventionaly done as an early event in the first generation (that is, this will be the first thing that happens in your simulation). For example:

1 early()  
{  
 sim.addSubpop("p1", 500);  
}

A few things to note here:

* The first line (1 early()) defines **when** this event occurs. The 1 indicates that this will happen in the first generation. The early() indicates that this is an early event in the generation (i.e. it happens before generation of offspring)
* The command sim.addSubpop("p1", 500); creates a subpopulation with the name "p1", which contains 500 individuals.

### 2.5.2 Events that occur every generation

The above sim.addSubpop() command was only executed in the first generation. What if we want an event that runs every generation? To do this, we simply create an event without specifying a generation number. For example, let’s say that in each generation, we want to print the list of all mutations that have reached fixation. We can do this with the following command:

late() {  
 sim.outputFixedMutations();  
}

In the first line of this command (late()), we do not specify the generation at which this happens. As a result, it will happen as a late event in *every* generation.

## 2.6 Ending a simulation

# About the Authors

These credits are based on our [course contributors table guidelines](https://www.ottrproject.org/more_features.html#giving-credits-to-contributors).

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Lead Content Instructor(s) | [FirstName LastName](link%20to%20personal%20website) |
| Lecturer(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | Delivered the course in some way - video or audio |
| Content Author(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | If any other authors besides lead instructor |
| Content Contributor(s) (include section name/link in parentheses) - make new line if more than one section involved | Wrote less than a chapter |
| Content Editor(s)/Reviewer(s) | Checked your content |
| Content Director(s) | Helped guide the content direction |
| Content Consultants (include chapter name/link in parentheses or word “General”) - make new line if more than one chapter involved | Gave high level advice on content |
| Acknowledgments | Gave small assistance to content but not to the level of consulting |
| **Production** |  |
| Content Publisher(s) | Helped with publishing platform |
| Content Publishing Reviewer(s) | Reviewed overall content and aesthetics on publishing platform |
| **Technical** |  |
| Course Publishing Engineer(s) | Helped with the code for the technical aspects related to the specific course generation |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/), [Ava Hoffman](https://www.avahoffman.com/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Ava Hoffman](https://www.avahoffman.com/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([ottrpal](https://github.com/jhudsl/ottrpal)) [Candace Savonen](https://www.cansavvy.com/), [John Muschelli](https://johnmuschelli.com/), [Carrie Wright](https://carriewright11.github.io/) |  |
| **Art and Design** |  |
| Illustrator(s) | Created graphics for the course |
| Figure Artist(s) | Created figures/plots for course |
| Videographer(s) | Filmed videos |
| Videography Editor(s) | Edited film |
| Audiographer(s) | Recorded audio |
| Audiography Editor(s) | Edited audio recordings |
| **Funding** |  |
| Funder(s) | Institution/individual who funded course including grant number |
| Funding Staff | Staff members who help with funding |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.5 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2023-10-31   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.5)   
## bookdown 0.24 2023-03-28 [1] Github (rstudio/bookdown@88bc4ea)   
## cachem 1.0.7 2023-02-24 [1] CRAN (R 4.0.2)   
## callr 3.5.0 2020-10-08 [1] RSPM (R 4.0.2)   
## cli 3.6.1 2023-03-23 [1] CRAN (R 4.0.2)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.20 2023-01-17 [1] CRAN (R 4.0.2)   
## fansi 0.4.1 2020-01-08 [1] RSPM (R 4.0.0)   
## fastmap 1.1.1 2023-02-24 [1] CRAN (R 4.0.2)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## glue 1.4.2 2020-08-27 [1] RSPM (R 4.0.5)   
## hms 0.5.3 2020-01-08 [1] RSPM (R 4.0.0)   
## htmltools 0.5.5 2023-03-23 [1] CRAN (R 4.0.2)   
## jsonlite 1.7.1 2020-09-07 [1] RSPM (R 4.0.2)   
## knitr 1.33 2023-03-28 [1] Github (yihui/knitr@a1052d1)   
## lattice 0.20-41 2020-04-02 [2] CRAN (R 4.0.2)   
## lifecycle 1.0.3 2022-10-07 [1] CRAN (R 4.0.2)   
## magrittr 2.0.3 2022-03-30 [1] CRAN (R 4.0.2)   
## Matrix 1.2-18 2019-11-27 [2] CRAN (R 4.0.2)   
## memoise 2.0.1 2021-11-26 [1] CRAN (R 4.0.2)   
## ottrpal 1.0.1 2023-03-28 [1] Github (jhudsl/ottrpal@151e412)   
## pillar 1.9.0 2023-03-22 [1] CRAN (R 4.0.2)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgconfig 2.0.3 2019-09-22 [1] RSPM (R 4.0.3)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## png 0.1-8 2022-11-29 [1] CRAN (R 4.0.2)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.4.0 2020-10-07 [1] RSPM (R 4.0.2)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## Rcpp 1.0.10 2023-01-22 [1] CRAN (R 4.0.2)   
## readr 1.4.0 2020-10-05 [1] RSPM (R 4.0.2)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## reticulate 1.28 2023-01-27 [1] CRAN (R 4.0.2)   
## rlang 1.1.0 2023-03-14 [1] CRAN (R 4.0.2)   
## rmarkdown 2.10 2023-03-28 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.3 2022-04-02 [1] CRAN (R 4.0.2)   
## rstudioapi 0.11 2020-02-07 [1] RSPM (R 4.0.0)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2023-03-28 [1] Github (R-lib/testthat@e99155a)   
## tibble 3.2.1 2023-03-20 [1] CRAN (R 4.0.2)   
## usethis 1.6.3 2020-09-17 [1] RSPM (R 4.0.2)   
## utf8 1.1.4 2018-05-24 [1] RSPM (R 4.0.3)   
## vctrs 0.6.1 2023-03-22 [1] CRAN (R 4.0.2)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2023-03-28 [1] Github (yihui/xfun@74c2a66)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library

# 3 References