SLiM Lecture Notes

We have seen how to come up with our own simulations, optimized them, and even use them for inference with abc

These are fine for many applications

However, as evolutionary biologists, we are often dealing with genetic processes
- These tend to be very complicated
- Lots steps, requiring a lot of code and optimization
- Luckily, Lots of work has been done to figure out how to best simulate these processes
- Goal of this lecture is to introduce you to arguably the most flexible software for simulating genetics slim

SLiM is a forward genetic simulator
- Forward means that it simulates events going forward in time
  - The way we normally think of them
  - As opposed to coalescent based simulators common for popgen
- Genetic means that it simulates explicit loci on a chromosome
- Simulator means that it is doing individual based modeling, rather than solving it analytically

The slim framework makes it incredibly flexible for simulating evolution

So what can you do with slim? You can almost do anything you can imagine!!

Examples from slim workshop

So how does one go about making a simulation in slim?

First of all, SLiM has a very nice graphical user interface for making and debugging sims
- The GUI has several different panels by default
  - Populations panel
  - Individuals panel
  - The chromosome panel
  - The scripting panel
  - The output panel
- The tick controls
- Show where to get help manual
- Show page to test eidos code
- Show debugging page and how to use it

Simulations are actually implemented with eidos, which is SLiMs own language
- Based off R, the majority of the basic functions are shared

- Show some eidos functions from manual using
    - rep('a',3);
    - rnorm(10);
- Major difference is every line is followed by ;

As stated before, slim is based off individual based modeling - everything is organized into hierarchical classes

- At its core, SLiM simulated individuals
    - Each individual has a genome
        - Within each genome, mutations occur
        - These mutations can be neutral or effect the fitness of an individual, though various ways
        - Genomes can have chromosomes with different genomic elements
            - exons, introns, etc
            - Different genomic elements can have different mutation types
        - Individuals can also be positioned in XYZ space
    - Individuals can be organized into subpopulations
        - Migration would occur between different subpopulations
    - Can also be arranged into a population
    - Populations make up a species
        - Species can interact
    - Different species make up a community

Generations in SLiM are split into different ticks. - time unit in simulation

- The ticks encode when certain actions should run within a generation
- These are first, early, and late
- Offspring generation happens between early and late ticks

Wright fisher vs non-wright fisher models

- A wright fisher model assumes non-overlapping generations and constant population size
- Most of the tutorials and things you would want to do are with wright fisher models
- Non-wright fisher relaxes this assumption
    - This allows you to model population growth and overlapping generations
    - Great for simulating more complex ecology

How do you learn how to use SLiM?

- First is to read the manual
    - very well documented, but not very searchable
    - Organized into a series of tutorials that cover almost everything that you might need to do in SLiM

Most helpful tool for looking up specific commands is the manual in SLiM GUI

Start with a recipe

    - All of the tutorials in the manual are in SLiM gui
    - Start there and modify to fit what you want to do

So lets play around with SLiM and see how it works

Well start with the simplest possible model - neutral evolution with random mating and constant population size

- Already this would take forever to code in python or R
- Just a few lines in SLiM

Starting from a blank SLiM script:

First step is initialize - all SLiM scripts have to start with this

```
initialize()
{
}
```

Then set mutation rate
`initializeMutationRate(1e-7);`

We will create just one mutation type, which will be neutral
`initializeMutationType("m1", 0.5, "f", 0.0);`

- m1 is the name
- 0.5 the dominance coeff - 0.5 is additive
- F is the distribution type to draw selection coefficients
    - F i flat
    - N is normal
    - E is exponential
    - Etc
- 0.0 is the selection coefficient

Then well initialize a genomic element type
`initializeGenomicElementType("g1", m1, 1.0);`

- G1 is the name
- M1 is the kind of mutation it can have
- 1 is the proportion of mutations of type m1

Then we initialize a chromosome with that genomic element type
`initializeGenomicElement(g1, 0, 99999);`

- G1 is genomic element type
- 0 is start
- 99999 is the end

Lastly well initialize the recombination rate
initializeRecombinationRate(1e-8);

Then we'll create our ticks

In the first tick we want to create the population before anything happens,
- Well use and early tick for that
- 1 early(){}

Then we will start a population with 500 individuals
sim.addSubpop("p1", 500);

Ok then we want to run our simulation for 1000 generations
The way to do that is just to tell SLiM that we want something done at generation 1000, and it will run through

```
10000 early()
{
	sim.simulationFinished();
}
```

Done!

Point out the mutation frequencies
Point out the individuals at the top
Play with some plots
Have it output some information
- p1.outputSample(10);

Show how to use the debug thing
- Add x = PI*2; and put a bug next to it

Modify to create balancing selection
- Add a second mutation type
- initializeMutationType("m2", 1.1, "n", 0.5, 0.1);
    - This one is called m2
    - It is overdominant, meaning the heterozygote has a higher fitness
    - Selection coefficient will be drawn from a normal distribution
    - Centered at 0.5, sd at 0.1
- Need to update the genomic element
    - initializeGenomicElementType("g1", c(m1,m2), c(1.0,0.001));
- We see that mutations are of frequency 0.5 - we made balancing selection
    - Easier to see in haplotype view
- Change recombination rate to 1e-6

- Now we dont get as much balancing selection because recombination breaks up haplotypes