SLiM Workshop Series #16: Separate Sexes

1. Start a new WF model in SLiMgui

Close all windows in SLiMgui. Then make a new WF model with command-N, or go to SLiMgui's File menu and choose **New**. Modify the code to look like this:

```
initialize() {
   initializeMutationRate(1e-7);
   initializeMutationType("m1", 0.5, "f", 0.0);
   initializeGenomicElementType("g1", m1, 1.0);
   initializeGenomicElement(g1, 0, 99999);
   initializeRecombinationRate(1e-8);
}
1 { sim.addSubpop("p1", 100); }
1e5 late() {
   subs = sim.substitutions;
   mt = mean(subs.fixationGeneration - subs.originGeneration);
   cat(subs.size() + " fixed mutations, " + mt + " mean time");
}
```

Besides stripping out comments, we've reduced the population size to `100`, increased the runtime markedly, and changed the final output event.

Recycle and run a couple of times, to get a good reading of the average behavior of the model. When the model finishes, it produces output like this (for three runs):

```
1057 fixed mutations, 392.51 mean time
```

```
984 fixed mutations, 387.218 mean time
```

```
1001 fixed mutations, 421.746 mean time
```

The variance observed between runs is pretty small, so you should see similar results.

2. Convert to a separate-sex model

The default WF model is hermaphroditic. Convert it to a model of separate sexes by adding this call to the top of the `initialize()` callback:

```
initializeSex("A");
```

The parameter `"A"` tells SLiM to model autosomes, rather than X or Y chromosomes (which we will see later). This is all that is needed; SLiM will now automatically track the sex of every individual, and will choose a female as the first parent and a male as the second parent in every biparental mating. Since WF models involve so much automatic behavior, there is no need to specify any of this.

Recycle and run the model. After several runs (you can do it just once if you like), it appears that modeling separate sexes has had little or no effect on the average behavior of the model:

```
998 fixed mutations, 395.082 mean time
```

```
983 fixed mutations, 389.035 mean time
```

```
1033 fixed mutations, 438.574 mean time
```

If there is any effect here, it is small; certainly not a factor-of-two shift.

3. Set a biased sex ratio

In WF models the sex ratio is an externally imposed parameter of the model. It is 0.5 by default, and WF models adhere to the specified sex ratio exactly: in each generation in this model there will be exactly 50 males and 50 females.

Let's change the sex ratio from this default of `0.5` to a strongly male-biased sex ratio. Change the `addSubpop()` call to supply a new value to the optional `sexRatio` parameter:

```
sim.addSubpop("p1", 100, 0.8);
```

In SLiM, sex ratio is always the proportion of males: M:(M+F), in other words, where M is the number of males and F is the number of females. A sex ratio of `0.0` is 100% female; a sex ratio of `1.0` is 100% male. (Be careful, as at least one other definition of sex ratio is popular in the literature.) We have set the population to be four-fifths male; that will mean exactly 80 males and 20 females in every generation, deterministically, since this is a WF model.

Running the model several times produces consistent results:

```
1018 fixed mutations, 267.35 mean time

1006 fixed mutations, 257.955 mean time

1001 fixed mutations, 258.508 mean time
```

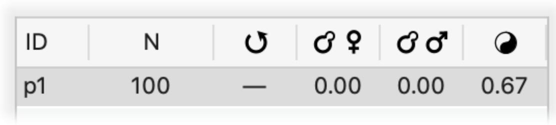
The mean number of fixed mutations remains the same, but they are fixing markedly faster.

4. Vary the sex ratio stochastically

Let's model variation in the sex ratio. Perhaps sex is determined by an environmental variable, such as temperature, that varies through time; or perhaps sex-specific diseases or fitness effects are causing sex-biased juvenile mortality; or whatever. To model this, add this event just after the generation 1 event (so that subpopulation `p1` exists by the time this code executes in generation 1):

```
early() {
   p1.setSexRatio(runif(1, 0.05, 0.95));
}
```

Recycle and run. The sex ratio is displayed in the population table in SLiMgui, incidentally:



ID	N	↻	♂♀	♂♂	◕
p1	100	—	0.00	0.00	0.67

The sex ratio is the rightmost column; the ratio is `0.67` in the generation shown.

Running this to completion a couple of times, we again get consistent results:

```
961 fixed mutations, 253.27 mean time

984 fixed mutations, 246.778 mean time

1033 fixed mutations, 234.256 mean time
```

As in the previous section, it appears that the mean number of fixed mutations is the same, but the fixation time is shorter than it was initially. This is because a biased sex ratio lowers the effective population size N_e, thereby increasing drift; the substitution rate of neutral mutations is independent of N_e, but fixation time is shorter with smaller N_e. You can see this effect in more detail if you open the "Fixation Time Histogram" window in SLiMgui for each of the models.

5. A sexual nonWF model

Let's shift gears. Close this model window and create a new nonWF model (shift-command-N, or **New (nonWF)** from the File menu). Stripped of comments, our initial model looks like this:

```
initialize() {
   initializeSLiMModelType("nonWF");
   defineConstant("K", 500);  // carrying capacity

   initializeMutationType("m1", 0.5, "f", 0.0);
   m1.convertToSubstitution = T;

   initializeGenomicElementType("g1", m1, 1.0);
   initializeGenomicElement(g1, 0, 99999);
   initializeMutationRate(1e-7);
   initializeRecombinationRate(1e-8);
}
reproduction() {
   subpop.addCrossed(individual, p1.sampleIndividuals(1));
}
1 early() {
   sim.addSubpop("p1", 10);
}
early() {
   p1.fitnessScaling = K / p1.individualCount;
}
2000 late() { sim.outputFixedMutations(); }
```

We're not going to try to compare the number of fixed mutations here to the WF model; it would be different, due to overlapping generations and so forth, and our goal here is not to show how to make a WF-style model with a nonWF SLiM model (there's a recipe in the manual for that). So let's just recycle and run to confirm that the model runs as it is.

6. Convert to a separate-sex model

As before, let's convert this model to use separate sexes. Insert this line just after the call to `initializeSLiMModelType()`:

```
initializeSex("A");
```

That should do it, right? Recycle and run. You will get an error:

```
ERROR (Subpopulation::ExecuteMethod_addCrossed): parent2 must be male in
sexual models (or hermaphroditic in non-sexual models).
```

In sexual models, for biparental matings the first parent must be female and the second parent must be male. In the WF model SLiM guaranteed this for us; in the nonWF model we must manage it ourselves. We can tweak the `sampleIndividuals()` call to choose males only:

```
subpop.addCrossed(individual, p1.sampleIndividuals(1, sex="M"));
```

Recycle and run, and you'll get another error:

```
ERROR (Subpopulation::ExecuteMethod_addCrossed): parent1 must be female in
sexual models (or hermaphroditic in non-sexual models).
```

We fixed the second parent, but the first parent can still be the wrong sex. In nonWF models, `reproduction()` callbacks are called, by default, for every individual in the population; but we can narrow that down by subpopulation or sex or both. Change the declaration of the `reproduction()` callback:

```
reproduction(NULL, "F") {
   subpop.addCrossed(individual, p1.sampleIndividuals(1, sex="M"));
}
```

The `NULL` tells SLiM that this callback applies to every subpopulation (if we had multiple subpopulations, we could specify `p1` or `p2` or whatever there, to have a narrower focus). The `"F"` tells SLiM that the callback should be called only with females as the focal individual. This will result in the first parent always being female; females are producing offspring, choosing males as their mates. This is appropriate for many sexual systems, but if you want to model males as the choosy sex in your system, just consider `"F"` to represent males and `"M"` to represent females – the designations are arbitrary, after all – or rewrite the callback as appropriate.

Change the final generation to `1e5`, so we can observe the model running for a little while, and then recycle and run. It should run without errors now. You can see in SLiMgui that the sex ratio is now varying stochastically around a mean of `0.5`; every generated child is randomly chosen as male or female (unlike in WF models), and so sometimes you get a couple more males or females.

7. Set a biased sex ratio

Let's change the sex ratio of the model, as we did with the WF model. Sex ratio is no longer an externally imposed parameter; it is emergent from the sexes of the offspring actually produced, which can depend upon anything in the model – chance, genetics, environment, time, or anything else. Let's follow the lead of the WF model, though, and model a scenario where there is simply an 80% probability that any given offspring will be male.

To do this, we can just pass that probability to `addCrossed()` directly:

```
reproduction(NULL, "F") {
   mate = p1.sampleIndividuals(1, sex="M");
   subpop.addCrossed(individual, mate, sex=0.8);
}
```

(The code is slightly restructured, for readability.) SLiM will choose the sex of the offspring based upon the probability given. We could also explicitly specify a sex of `"M"` or `"F"` for the offspring.

EXERCISE: Try changing the sex ratio to `0.99`. Recycle and run. What happens? Why? What do you think about this problem?

EXERCISE: The next step for the WF model was varying the sex ratio stochastically. Adapt this nonWF model to do the same. A target sex ratio should be drawn from [`0.3`, `0.7`] once per generation and used for all reproduction. You should know how to do this; hints follow.

Hint 1: When you draw the target ratio, where should you store it for future reference? Remember `setValue()`? Perhaps `sim.setValue()` would prove a convenient spot.

Hint 2: When should the target ratio be drawn? Review the generation life cycle for nonWF models. If you want something to happen before offspring generation, when in the generation cycle does it need to happen? An `early()` event is too late; it runs after offspring generation has completed. How about a `late()` event in the previous generation, to run just before reproduction occurs at the beginning of the following generation?

EXERCISE: There is actually another way to make something run immediately before offspring generation in a nonWF model. Declare a second `reproduction()` callback, *above* the other callback. In it, do the sex ratio draw; then execute `self.active = 0;`. Try it. Why does it work?

8. BEEP! With extra time, look at section 6.2.3, on modeling sex chromosome evolution.