SLiM Workshop Series #17: Modeling Haploidy

1. Start a new WF model in SLiMgui

Close all windows in SLiMgui. We're going to start with SLiMgui's default WF model. To get that, press command-N, or go to SLiMgui's File menu and choose **New**.

We need to make a few small changes. Extend the final generation to `20000` for some extra runtime. Then change the dominance coefficient for `m1` from `0.5` to `1.0`:

```
initializeMutationType("m1", 1.0, "f", 0.0);
```

Since we're making a haploid model, there is no such thing as dominance, and we want mutations to have their full selection effect when present in a single genome. For `m1` here it doesn't actually matter, though, since `m1` is neutral anyway.

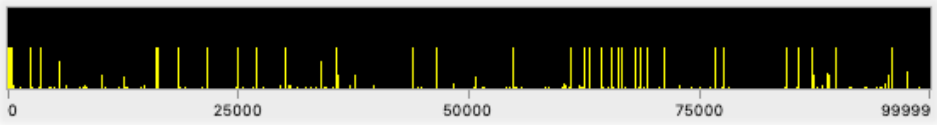
Next, change the recombination rate to zero:

```
initializeRecombinationRate(0);
```

Remove the two output events that call `outputSample()`, just for simplicity. Finally, change the model to reproduce clonally by adding a call to `setCloningRate()` just after `p1` is created:

```
p1.setCloningRate(1.0);
```

If you recycle and run at this point, you will see some odd behavior:



Mutations reach a frequency of `0.5` and then stop; nothing fixes. At this point we have a model of clonal diploids with no recombination at all. Each individual simply inherits its two genomes from its parent, plus new mutations; if a new mutation arises in one genome, it has no way to ever get into the other genome and so frequencies above `0.5` will never occur. Clonal reproduction is weird. But in any case we don't want to be modeling diploids, so don't worry about this issue; it will go away in the next step.

2. Switch to haploidy

To convert this to a model of haploids, we need to do two things: remove new mutations added to the second genome of individuals (which doesn't "really" exist, as far as we are concerned), and cause fixation to occur at a frequency of `0.5`. We can do both with a new `late()` event that runs in every generation:

```
late() {
   // remove any new mutations added to the disabled diploid genomes
   sim.subpopulations.individuals.genome2.removeMutations();

   // remove mutations in the haploid genomes that have fixed
   muts = sim.mutationsOfType(m1);
   freqs = sim.mutationFrequencies(NULL, muts);
   if (any(freqs == 0.5))
      sim.subpopulations.genomes.removeMutations(muts[freqs == 0.5], T);
}
```

The first part removes new mutations added to second genomes. It simply gets a vector of all of the second genomes in the model, with `sim.subpopulations.individuals.genome2`, and calls `removeMutations()` on that vector to remove all mutations from all of them. SLiM will still added new mutations to them in each generation – in WF models we have no way to tell it not to – but we just remove them again immediately, and the overhead is actually quite small.

The second part handles the fixation of mutations. Mutations in this haploid model will reach a population-wide frequency of `0.5` when they fix, because they will not be present in the second genomes of individuals. We have no way to tell SLiM to use a different threshold for fixation, but we can do it ourselves in script quite easily. First we get a vector of all `m1` mutations in the model, and then fetch the frequencies for those mutations. If there are any that have reached a frequency of `0.5` (using the `any()` function – remember it? – to test for any `T` value in the `logical` vector generated by `freqs == 0.5`), then we call `removeMutations()` on all genomes to remove the fixed mutations globally. Passing `T` to `removeMutations()` tells SLiM to convert the removed mutations into `Substitution` objects for us, so it is just as if they had been substituted by SLiM naturally.

Recycle and run the model, and you will now see mutations fixing when they reach half the height of the chromosome view. That's all we need to do; we now have a model of clonal haploids.

3. Start with a new nonWF model in SLiMgui

Leave that window open for comparison. Now create a new default nonWF model with shift-command-N or by choosing **New (nonWF)** from the **File** menu.

As before, change the dominance coefficient for `m1` to `1.0` (although again it doesn't really matter since this is a neutral model). You can remove the line:

```
m1.convertToSubstitution = T;
```

since mutations will never reach a frequency of `1.0` anyway; SLiM will never automatically detect fixation, and we will have to do it ourselves, as before. Also, set the recombination rate to `0` again, and extend the final generation to `20000` again.

Finally, change the `reproduction()` callback to use clonal reproduction:

```
reproduction() {
   subpop.addCloned(individual);
}
```

Each focal individual reproduces itself once in each generation by generating a clone. This is, as before, a model of clonal diploids, and if you recycle and run it you will see the same behavior as before: mutations reaching a frequency of 0.5 and sticking there without fixing.

4. Switch to haploidy

As before, we will now switch the model to haploidy. The first step is to replace the `reproduction()` callback again, with this code:

```
reproduction() {
   subpop.addRecombinant(genome1, NULL, NULL, NULL, NULL, NULL);
}
```

The `addRecombinant()` method is a special reproduction method that lets you very precisely control the details of offspring generation. With these particular parameters, we tell it that the first offspring genome should be a clone of the first genome of the focal parent (the pseudo-parameter `genome1`), while the second offspring genome should be kept empty (without even new mutations added to it). You can read more about the details of `addRecombinant()` in the online help, as usual. It is quite useful for building models of unusual mating systems; it could also be used to implement

horizontal gene transfer in a haploid clonal model, for example (there's a recipe in the manual for that), or a model of haplodiploid reproduction such as exists in social insects.

That solves the problem of removing new mutations in the second genomes of individuals, since such mutations won't be generated in the first place – a cleaner design that the WF model. Now we just need to substitute any mutations that have fixed with a frequency of 0.5. We do that with the same code as before:

```
late() {
   muts = sim.mutationsOfType(m1);
   freqs = sim.mutationFrequencies(NULL, muts);
   if (any(freqs == 0.5))
      sim.subpopulations.genomes.removeMutations(muts[freqs == 0.5], T);
}
```

If you recycle and run, you should see much the same haploid clonal behavior as before, with fixation occurring at a frequency of `0.5`.

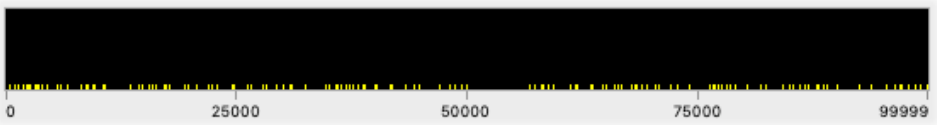
5. Adding non-overlapping generations

You might have noticed that mutations seem to take longer to fix in the nonWF model. This is because it has overlapping generations and age structure, unlike the WF model. The mean lifetime is longer, and so the time to fixation in "generations" is longer. In some cases this might be desirable; bacteria, for example, are clonal haploids that exhibit overlapping generations, and the nonWF model will capture those dynamics automatically. In other cases, however, such as many clonally reproducing plants, non-overlapping generations would be better.

This is easy to change. For starters, replace the `early()` event that generates density-dependent selection with a new `early()` event:

```
early() {
   inds = sim.subpopulations.individuals;
   inds[inds.age > 0].fitnessScaling = 0.0;
}
```

If you recycle and run, you will see that the model behaves a bit oddly. The population size stays at exactly `10` instead of growing to `K`, and mutations seem to get blocked at a very low frequency:



If you think about it, this makes sense. The `reproduction()` callback is designed to generate exactly one clonal offspring from each parent, and then the parents are killed off. Since the model starts with `10` individuals, the next generation also has `10`, and the next; it never grows. There is also no opportunity for clonal competition to occur, because every offspring that is generated survives to generate an offspring of its own. If effect, we have ten completely independent lineages, each containing one individual in every generation, and apart from new mutations nothing changes.

EXERCISE: Implement a solution to this problem that allows the population size to grow up to `K` and then equilibrate there. You will probably want the `reproduction()` callback to generate more than one offspring (allowing the population to grow), and to bring back density-dependent selection.

6. BEEP! With extra time, look at section 16.14 in the SLiM manual, which shows how to model horizontal gene transfer in a bacterial model with a circular chromosome.