14.5 Modeling both X and Y Chromosomes with a Pseudo-Autosomal Region (PAR)

SLiM has built-in support for modeling either the X or Y chromosome when sex is enabled (see section 8.3.4). However, some models need to go beyond this built-in support. You might wish to model *both* the X and Y chromosomes, and you might even wish to model a pseudo-autosomal region (PAR) – a region in which recombination between the X and Y occurs freely, giving the region evolutionary dynamics similar to those of an autosome. Both males and females are diploid for genes in the PAR; females have two copies of the PAR on their two X chromosomes, whereas males have the same PAR on their X, and a homologous PAR on their Y. Because crossing over occurs freely between the X and Y within the PAR, genes in the PAR exhibit an autosomal pattern of inheritance rather than sex-linked inheritance.

Modeling this in SLiM is possible by implementing your own sex-chromosome mechanics, which is fairly straightforward. In this section we'll explore a simple model of neutral X and Y chromosome evolution with a single PAR connecting them. This model was provided by Melissa Jane Hubisz, and has been adapted for publication as a recipe here. The recipe:

```
initialize()
{
    initializeMutationRate(1.5e-8);
    initializeMutationType("m1", 0.5, "f", 0.0);     // PAR
    initializeMutationType("m2", 0.5, "f", 0.0);     // non-PAR
    initializeMutationType("m3", 1.0, "f", 0.0);     // Y marker

    // 6 Mb chromosome; the PAR is 2.7 Mb at the start
    initializeGenomicElementType("g1", m1, 1.0);     // PAR: m1 only
    initializeGenomicElementType("g2", m2, 1.0);     // non-PAR: m2 only
    initializeGenomicElement(g1, 0, 2699999);        // PAR
    initializeGenomicElement(g2, 2700000, 5999999);  // non-PAR

    // turn on sex and model as an autosome
    initializeSex("A");

    // no recombination in males outside PAR
    initializeRecombinationRate(c(1e-8, 0), c(2699999, 5999999), sex="M");
    initializeRecombinationRate(1e-8, sex="F");
}
1 late() {
    // initialize the pop, with a Y marker for each male
    sim.addSubpop("p1", 1000);
    i = p1.individuals;
    i[i.sex == "M"].haploidGenome2.addNewMutation(m3, 0.0, 5999999);
}
modifyChild() {
    numY = sum(child.haplosomes.containsMarkerMutation(m3, 5999999));

    // no individual should have more than one Y
    if (numY > 1)
       stop("### ERROR: got too many Ys");

    // females should have 0 Y's
    if (child.sex == "F" & numY > 0)
       return F;

    // males should have 1 Y
    if (child.sex == "M" & numY == 0)
       return F;
```

```
        return T;
    }
    10000 late() {
        p1.outputMSSample(10, replace=F, requestedSex="F");
    }
```

The `initialize()` callback sets up the genetic structure. This model turns on sex, because we want SLiM to track males and females for us, but it requests modeling of an autosome with `initializeSex("A")`; we will handle the tracking of the X versus Y chromosomes ourselves. To do that, we set up a special "marker" mutation type, `m3`, that will be used to tag Y chromosomes, as we will see shortly. We set up a 6 Mb chromosome with a 2.7 Mb PAR at the beginning; the PAR uses mutation type `m1` (through genomic element type `g1`), while the rest of the chromosome uses mutation type `m2` (through genomic element type `g2`), so that we can easily tell sex-linked mutations from pseudo-autosomal mutations later on. The only wrinkle during initialization is that we set up a recombination map in males that prevents recombination outside the PAR; females, which have two X chromosomes, are allowed to recombine freely.

Next we have a tick `1 late()` event that sets up the initial population. After making a new subpopulation in the usual way with `addSubpop()`, it gets the male individuals, selects only their second `Haplosome` objects using the `haploidGenome2` property, and adds an `m3` mutation to them to mark them as Y chromosomes. These marker mutations will be handled in the usual way by SLiM, so they will be inherited and will continue to mark Y chromosomes in future generations. Because recombination is prevented in males outside the PAR, they will stay associated with the non-PAR Y chromosome genetic information.

There is one problem with this scheme, however. When SLiM generates offspring, it has its own ideas about whether a given child ought to be male or female. We need to follow SLiM's guidance on this, otherwise our model will end up with individuals that SLiM considers female but that possess a Y chromosome, and individuals SLiM thinks are male but that have no Y. This is the purpose of the `modifyChild()` callback. It simply compares what SLiM expects for the sex of the child (`child.sex`) with the genetics that SLiM is proposing that the child will inherit (`child.haplosomes`). If they don't match, then it returns F to indicate that SLiM needs to choose new parents and try again. Note the `containsMarkerMutation()` method; this just checks for a mutation of a given type at a given position, which can be done quite quickly by SLiM. It is thus optimal when the position of a mutation (if it exists) is already known, as it is here.

Because `modifyChild()` callbacks get called quite frequently (once for each new offspring generated by SLiM, or even more in this case since the callback rejects some proposed children), the speed of callback code is essential. The callback above has very clear logic, but it is slow in several ways. It does a safety check that is never, in fact, hit (since the model works properly); it assigns a value into a variable, `numY`, which is relatively slow because it requires Eidos to set up a symbol table entry; and it performs some unnecessary logical operations and tests (if `child.sex` is not `"F"` then it *must* be `"M"`, for example). Worst of all, it calls `containsMarkerMutation()` for both child haplosomes, but this is completely unnecessary because SLiM always guarantees (unless you change it in script somehow) that an individual's first haplosome is inherited from the maternal parent and its second haplosome is inherited from the paternal parent. If a Y chromosome is present, it will therefore always be in the second haplosome; there is no need to check the first chromosome for having a Y chromosome. With an optimized version of the callback, the model runs about twice as fast, a very significant difference. The optimized callback:

```
    modifyChild() {
        // females should not have a Y, males should have a Y
        if (child.sex == "F")
          return !child.haploidGenome2.containsMarkerMutation(m3, 5999999);
```

```
      else
         return child.haploidGenome2.containsMarkerMutation(m3, 5999999);
   }
```

This will produce the same result in all cases (unless the model has already broken, such as by an individual having two Y chromosomes, or by a Y chromosome being present in the first haplosome of a proposed child somehow).

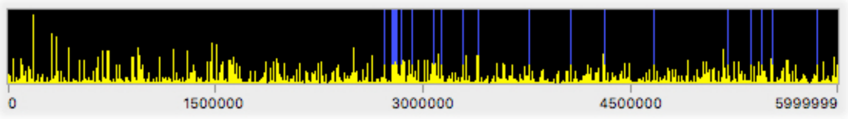
Finally, we have a `late()` event that outputs an MS-format sample from the females in the population; among other things, this establishes the end of the model as tick `10000`.

There is one remaining issue. Addressing it is optional, in a sense, but if we don't address it the model will run slower and slower until it grinds nearly to a halt. The problem is that while PAR mutations will fix and be converted into `Substitution` objects automatically by SLiM as usual, non-PAR mutations will not. This is because their threshold for fixation is lower than SLiM realizes; only a quarter of `Haplosome` objects are Y chromosomes, and only three-quarters are X chromosomes, so sex-linked mutations need to fix when they reach those frequencies, not a frequency of `1.0` as SLiM expects. Fixing this is not difficult, but it does require a bit of code:

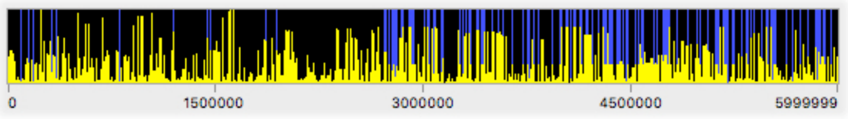
```
1:10000 late() {
   // periodically remove m2 (non-PAR) mutations that are fixed in X or Y
   // m1 (PAR) mutations will be automatically removed when fixed
   if (sim.cycle % 1000 == 0) {
      numY = sum(p1.individuals.sex == "M");
      numX = 2 * size(p1.individuals) - numY;

      // look at the mutations in a single Y chromosome
      // to find mutations that are fixed in all Y's
      firstMale = p1.individuals[p1.individuals.sex == "M"][0];
      fMG = firstMale.haplosomes;

      if (fMG[0].containsMarkerMutation(m3, 5999999)) {
         firstY = fMG[0];
         firstX = fMG[1];
      } else if (fMG[1].containsMarkerMutation(m3, 5999999)) {
         firstY = fMG[1];
         firstX = fMG[0];
      } else
         stop("### ERROR: no Ys in first male");

      ymuts = firstY.mutationsOfType(m2);
      ycounts = sim.mutationCounts(NULL, ymuts);
      removeY = ymuts[ycounts == numY];

      // now do the same for the X
      xmuts = firstX.mutationsOfType(m2);
      xcounts = sim.mutationCounts(NULL, xmuts);
      removeX = xmuts[xcounts == numX];

      cat("Cycle " + sim.cycle + ": Removing ");
      cat(removeX.size() + "/" + removeY.size() + " on X/Y\n");

      removes = c(removeY, removeX);
      sim.subpopulations.haplosomes.removeMutations(removes, T);
   }
}
```

This event should ideally be inserted before the tick `10000 late()` event that produces final output. It runs every `1000` ticks, since the operation it performs is somewhat time-consuming; doing it every tick would be wasteful. It finds a male individual, and uses that individual as a template for finding and removing fixed sex-linked mutations. Any mutation that has fixed must be possessed by the template male (by the definition of "fixation"), so the code just gets all the mutations of type `m2` (non-PAR mutations) from the male's X and Y and checks them all for fixation. The fixation check itself is done using `mutationCounts()`, which returns the number of occurrences of the mutations – population-wide, in this case, because of the `NULL` passed for the first parameter. The event prints the number of X-linked and Y-linked mutations that it intends to remove, and then it removes them with `removeMutations()`. The optional `T` value passed for the second argument to `removeMutations()` (which is named `substitute`) indicates that SLiM should create `Substitution` objects for the removed mutations; we are notifying SLiM that those mutations are in fact fixed, even though SLiM doesn't realize it. If you don't care about that record-keeping, you can omit that optional `T` value, and the mutations will simply be removed.

And that's it. We've implemented tracking of Y chromosomes with marker mutations, we've guaranteed that those markers stay correctly synchronized with offspring sex, and we've added machinery to detect fixed sex-linked mutations and turn them into Substitutions just as SLiM does for autosomal mutations. If we run this model in SLiMgui with display of fixed mutations turned on (with the ✱ button to the right), the behavior of the PAR versus the non-PAR regions is easy to see as soon as the first pass of the fixation check has run in tick `1000`:



The PAR, on the left, behaves as an autosome, so it takes a while for mutations to fix; one is close to fixation, but none has made it there yet. The non-PAR region, on the right, behaves as separate sex chromosomes that cannot recombine, and each sex chromosome is present in fewer copies than the PAR; mutations in that region thus have a smaller effective population size, and fix more rapidly. The Y, present in only a quarter as many copies as the PAR, fixes particularly quickly; all 20 of the fixations here are in fact on the Y. After tick `3000`, the situation is similar:



There have now been 19 fixations on the X, 120 on the Y, and only 11 in the PAR. The trend is clear, and it appears that our model of a pseudo-autosomal region is functioning as expected.