SLiM Workshop Series #6: Recombination

1. Start a new model in SLiMgui

Run SLiMgui, and close any existing model windows. Press command-N to make a new model window. We will work with this model as a starting point.

This initial model defines a single uniform recombination rate along the whole chromosome:

```
initializeRecombinationRate(1e-8);
```

This tells SLiM that the probability of recombination between any given base position and the next base position is 10^{-8}, in each new genome being generated by meiosis, in each generation.

2. Set up a simple recombination map

We want to extend this model to define a simple recombination map that has a high rate at one end and a low rate at the other end. Click inside the parentheses of `initializeRecombinationRate()` and look at the function prototype shown at the bottom of SLiMgui's window:

```
(void)initializeRecombinationRate(numeric rates, [Ni ends = NULL],
   [string$ sex = "*"])
```

Thus far we have just been passing a singleton `float` for rates, and using the default values for the other parameters. We will not use the `sex` parameter here – it is for setting up separate recombination maps for males versus females – but we do now need to use `ends`. This parameter is used to designate the *last* base position for each recombination rate specified in `rates`. It is easier to show this than to explain it, so replace the previous call with:

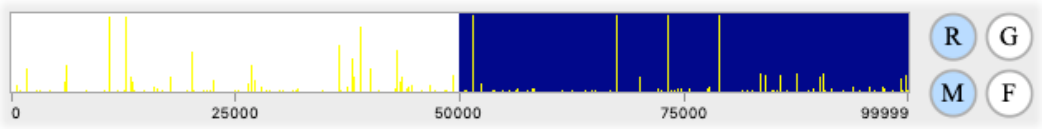
```
initializeRecombinationRate(c(1e-5, 0), c(49999, 99999));
```

Also change the generation for the final output event to `200000`, so that we can observe the model for a longer time (you don't have to run all the way to the end each time, of course). Recycle and run, and observe the model closely. The difference is subtle, but you may be able to see that in the left half of the chromosome mutations are drifting independently – their bars rise and fall individually – whereas in the right half of the chromosome mutations exhibit linkage into haplotypes that rise and fall in synchrony, as if their bars were connected.

Note how these values are interpreted. The rate `1e-5` is used as the probability of a crossover between each pair of bases from the pair (`0`, `1`) to the pair (`49998`, `49999`); the last use of the rate `1e-5` is to the *left* of base `49999`. After that, the rate `0` is used from pair (`49999`, `50000`) to the final pair, (`99998`, `99999`).

3. Visualize the recombination map in SLiMgui

It would be nice to be able to see the recombination map represented graphically, and SLiM can do this for you. Click the circular button labeled **R**, to the right of the chromosome view, and you should see something like this:



Colors used by SLiMgui for this display range from white (very high recombination rates) through dark blue to black (very low recombination rates). The scale is somewhat arbitrary and the display is more qualitative than quantitative, but it can at least provide a sense of whether the map is defined as intended.

EXERCISE: Change the recombination rate map in this model to define three regions instead of two. Use an intermediate rate of `1e-7` for the new region. Recycle and run to confirm visually that you have defined it as you intended.

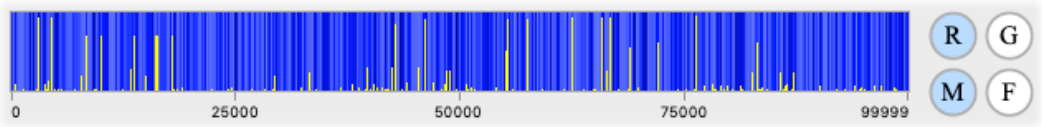
4. Define a random recombination rate map

Suppose we want to model a particular species – one we study in a real-world system, say. We have a sense of the general range of recombination rates that our species has along the chromosome, but a detailed recombination rate map has not yet been constructed for our species of interest. We'd like to know whether the variable recombination rate along the chromosome is likely to influence the results our model generates, by comparing a model with a random map to a model with a uniform recombination rate. We can set up a random recombination rate map very easily; just replace the previous call to `initializeRecombinationRate()` with this code:

```
rates = runif(1000, 1e-9, 1e-7);
ends = c(sort(sample(0:99998, 999)), 99999);
initializeRecombinationRate(rates, ends);
```

We will model a chromosome with 1000 recombination regions of different rates, with random lengths tiled across the chromosome. The `runif()` call generates `1000` random draws from a uniform distribution within the interval [`1e-9`, `1e-7`]. Then we generate 999 randomly chosen base positions in the range `0:99998` using `sample()`; since that samples without replacement by default, the base positions will all be different. We use `sort()` to sort them into ascending order, and append the final base position for the chromosome, `99999`, to the end of the vector. The end result is a vector 1000 elements long that provides the end positions for each recombination rate, ending at the end of the chromosome.

Recycle and run this model with recombination rate map display enabled to see something like this:



The hotspots and coldspots defined by the random recombination rate map can be seen clearly. Of course this example is a bit artificial; you would likely want to use a more realistic distribution for both the rates and the endpoints. Nevertheless, it illustrates the concept. Later in this exercise we will see how to read and use an empirically measured recombination rate map from a file on disk.

EXERCISE: Change the maximum rate from `1e-7` to `1e-6`, and recycle and run. Note that the map is mostly white; most of the rates drawn will be at the top end of the range, because the interval [`1e-7`, `1e-6`] is almost ten times as wide as the interval [`1e-9`, `1e-7`].

EXERCISE: Given that observation, it might be nice to draw rates on a logarithmic scale instead, so that the low end of the range is well-represented. To do this, draw exponents using `runif()` with a min of `-9` and max of `-6`, and then raise `10` to the power of each exponent; remember that `^` is the exponentiation operator in Eidos. Recycle and run, and observe the difference.

5. Set up a two-chromosome model

SLiM "officially" models only a single chromosome, but it is easy to get it to model more than one "effective chromosome" with a custom recombination map. The trick is that a recombination rate of `0.5` between two bases produces a free recombination point; the region to the left of the break and the region to the right will assort independently during meiosis, since a crossover will be drawn between them 50% of the time.

Let's convert our model to define two effective chromosomes, by setting up a map with a rate 0.5 point between two specific bases. Replace the previous recombination rate code with this:

```
initializeRecombinationRate(c(1e-10, 0.5, 1e-10), c(39999, 40000, 99999));
```

That's all it takes. Recycle and step once, and you should see a recombination rate map like this:

The white bar at position 40000 indicates the position of the break between the two effective chromosomes. If you run this model, you might be able to see linked haplotypes rising and falling together in each of the two effective chromosomes, but unlinked between the chromosomes because of the independent assortment of the chromosomes. (If that's hard to see, try changing the `1e-10` rates to `0`, and increasing the population size to something like `2000`.)

There is nothing magical about this; it's just a particular recombination map. You can use this technique with any recombination rates within the chromosomes, and with any genomic structure as defined by mutation types, genomic element types, and genomic elements. In particular, the structure defined by genomic elements is entirely orthogonal to the structure defined by the recombination rate map; the two do not need to correspond at all.

EXERCISE: Modify this recipe to have three effective chromosomes instead of two, placing the additional break wherever you wish. Recycle and step to verify that the map looks as it ought to.

EXERCISE: Modify this recipe so that, instead of modeling a chromosome or chromosomes as such, it models 1000 unlinked loci, each a single base position in length. This should be very easy – it is actually a simplification of the code as it stands. A hint follows.

Hint: If the recombination rate is uniform along the chromosome, there are no recombination regions and no endpoints for those regions; there is just a rate.

6. Read a recombination rate map from a file

As a final example of working with recombination rate maps, we'll look at loading a rate map from a file. Close the current SLiMgui window and open recipe 6.1.2 from the **File** menu, under **Open Recipe**. For this recipe to run, you will need to download the input file it uses, which is a *Drosophila* recombination rate map published in Comeron et al. 2012. There is a URL given in the recipe's script for the Recipes zip archive (also linked to from the SLiM home page); copy and paste that URL into a web browser to download the archive, then double-click the archive to unzip it, locate the file inside the unzipped folder, and copy it to the Desktop where the model expects to find it.

The core code that reads and interprets the recombination rate map file is:

```
lines = readFile("~/Desktop/Comeron_100kb_chr2L.txt");
rates = NULL;
ends = NULL;

for (line in lines) {
   components = strsplit(line, "\t");
   ends = c(ends, asInteger(components[0]));
   rates = c(rates, asFloat(components[1]));
}

ends = c(ends[1:(size(ends)-1)] - 2, 23011543);
rates = rates * 1e-8;
initializeRecombinationRate(rates, ends);
```

First, `readFile()` reads in the entire file to produce a `string` vector, with one element per file line. Then the code loops through the lines one by one. The lines in the file look like this:

```
1        0
100001   0
200001   0
300001   0.234550139
400001   0.234550139
500001   1.993676178
...
```

Those six lines would be the first six `string` elements in the vector `lines`. Let's consider what the loop does for the last of those six lines. The value of `line`, for that iteration of the `for` loop, would be `"500001\t1.993676178"`, where `\t` represents a tab character. The `strsplit()` function splits that string by tab characters (since `"\t"` is passed to it as the separator), producing a vector with two string elements, `"500001"` and `"1.993676178"`. The first of those strings, `components[0]`, is converted to an `integer` and appended onto the end of the variable ends with `c()`; the second, `components[1]`, is converted to a `float` and appended to the end of `rates`.

The positions specified in the file are starts, not ends, and they are one-based (the first base position on the chromosome is position 1), whereas SLiM uses zero-based positions. The `ends` vector therefore needs to be adjusted by removing the first value (the start position of `1`, which we do not need) and adding the position of the end of the chromosome (`23011543`). The subtraction of `2` switches from one-based to zero-based positions, and also from starts to ends, each of which entails a shift by one. Finally, the rates in the file are in units of cM/Mbp and require conversion.

Recycle and run the model, and turn on display of the recombination rate map with the **R** button, as before; if all is well, you should see the *Drosophila* 2L recombination rate map! Incidentally, separate recombination rate maps can be set for males and females with the `sex` parameter to `initializeRecombinationRate()`, so setting up sex-specific rates (as exist in *Drosophila*) is easy.

This recipe provides a general strategy for reading text files in Eidos: read in the lines of the file with `readFile()`, loop through them with a `for` loop, use `strsplit()` to split tab-separated or comma-separated values into separate strings, and handle those values however you wish. The SLiM-Extras repository at https://github.com/MesserLab/SLiM-Extras has example code for a function named `readIntTable()` that reads in a two-dimensional matrix of values, providing further possibilities for writing file-driven models that encapsulate specific behavior, such as migration rates between subpopulations, in external files.

7. BEEP! With extra time, read manual section 1.5.6, about the "DSB" recombination model.