Inferring demography from tracts of identity by state

Kelley Harris
kharris@math.berkeley.edu

Contents

This manual describes how to extract tracts of identity by state (IBS) from DNA sequence data and use them for demographic inference. For conceptual background on the method, see the manuscript "Inferring demographic history from a spectrum of shared haplotype lengths" published in *PLoS Genetics* by Harris and Nielsen in 2013.

1 Formatting input data

1.1 Creating a popdata file for each population

To extract length distributions of IBS tracts from your data, you first need to create a plain text file for each population that contains four tab-delimited columns with the following data:

Chromosome	Site	Reference allele	Sample configuration
1	41342	T	TTTAATTTTTATTTATTAT
1	41791	A	AAAAAAAAGGAAAAGAAAA

Each column of the 'sample configuration' block contains genotype information from a single haplotype in your sample, specifying its allele at each listed segregating site. If your data are available as a VCF file, you can convert it into this `popdata` format using the following `vcf-query` command from the software package `vcf-tools`:

```
vcf-query file.vcf -f '%CHROM\t%POS\t%REF\t[%GT]\n'
```

When inferring the joint history of two populations, we strongly recommend using data produced on a single sequencing pipeline where SNPs are called jointly across populations using a tool such as the GATK Unified Genotyper.

Scaffolds vs. true chromosomes: Depending on the nature of your data, you can substitute arbitrary scaffold or "pseudochromosome" numbers for chromosome numbers. This should not affect the program performance. However, chromosome numbers and site numbers should appear in numerical order. Chromosome designations must be numbers, not other strings like 'X' or 'Y'.

1.2 Data parsing pipeline

This section gives the sequence of python scripts you will need to call to extract IBS tract length distributions from a collection of single-population files in preparation for demographic inference.

If all of your popdata files have the same number of lines representing the same set of segregating sites, you can skip step 1 and move to step 2 of the pipeline. If not, step 1 describes how to create compatible files that have the same lists of variable sites.

1. compatibilize_hapfiles.py

   Short explanation: This script takes two required arguments pop1 and pop2 and can take two optional arguments label1 and label2. If you want to look at IBS tracts shared between population 1 and population 2, represented by the popdata sample files pop1.popdata and pop2.popdata, use the command

   ```
   python compatibilize_hapfiles.py pop1 pop2
   ```

   to generate two altered sample files pop1_for_pop2.popdata and

   pop2_for_pop1.popdata. If your filenames pop1 and pop2 are very long, you can enter optional abbreviated names label1 and label2 and the command

   ```
   python compatibilize_hapfiles.py pop1 pop2 label1 label2
   ```

   so that your output files will be named label1_for_label2.popdata and label2_for_label1.popdata instead.

2. parse_between_pops_allpairs.py

   This script reads in two compatibilized popdata files label1.popdata and label2.popdata and generates an output file label1_vs_label2.ibs with the length distribution of IBS tracts shared between the two input haplotype samples. The input files are required to have the same number of lines corresponding to the same list of sites, as accomplished by the compatibilize_hapfiles.py script.

If $h_0, \ldots, h_{n-1}$ are the haplotypes from population 1 and $g_0, \ldots, g_{m-1}$ are the haplotypes from population 2, then by default, the output distribution pools all nm possible comparisons of a haplotype h_i with a haplotype h_j. This is accomplished by the following command:

```
python parse_between_pops_allpairs.py label1 label2 MaskFolder
        percentMissing
```

The arguments `MaskFolder` and `percentMissing` are used to filter out regions of low-quality data and will be explained later. If you do not wish to filter the data, set `MaskFolder` equal to `None` and `percentMissing` equal to 1:

```
python parse_between_pops_allpairs.py label1 label2 None 1
```

If you wish to use only a subset of the haplotype samples from your `popdata` files, you can specify optional arguments `start1`, `end1`, `start2`, and `end2`:

```
python parse_between_pops_allpairs.py label1 label2  MaskFolder
                         percentMissing start1 end1 start2 end2
```

If you set `start1`, `end1`, `start2`, and `end2` equal to i, j, k and l, respectively, subject to the constraints $0 \leq i < j < n$ and $0 \leq k < l < m$, this restricts your analysis to the haplotypes $h_i, \ldots, h_{j-1}$ from population 1 and $g_k, \ldots, g_{l-1}$ from population 2.

The filtering argument `percentMissing` can be set to a threshold value t between 0 and 1. The script then throws out all IBS tracts with a frequency of 'N's (missing data) greater than t. See Section 1.3 on data filtering and quality control for more information about this and about specifying a mask folder.

In addition to the final output file `label1_vs_label2.ibs`, the `parse_between_pops_allpairs.py` script generates several intermediate files: `label1_vs_label2_lengths_unsorted.txt`, `label1_vs_label2_position_info.txt`, `label1_vs_label2_position_info_sorted.txt`, and `label1_vs_label2_sorted.txt`. These can often be ignored and/or overwritten, but the `label1_vs_label2_position_info_sorted.txt` file can be useful for locating regions with clustered long IBS tracts that might be artifacts of bioinformatical errors.

3. `parse_between_pairs_noreuse.py`

   This script can be run instead of `parse_between_pairs_allpairs.py` to generate a list of IBS tract shared between populations:

   ```
   python parse_between_pops_allpairs.py label1 label2  MaskFolder
                                percentMissing start1 end1
   ```

   Instead of sampling all mn possible haplotype pairs, it generates IBS tracts shared between haplotype i from population 1 and haplotype i from population 2 for all i ranging from `start1` to `end1`. It therefore runs faster and generates a smaller set of IBS tract lengths, but these lengths are less correlated with each other than the lengths output by `parse_between_pairs_allpairs.py`.

4. `parse_within_pop_allpairs.py`

   This script reads in a single `popdata` file `sample.popdata` and generates an output file `sample.ibs` with the length distribution of IBS tracts shared within pairs of haplotypes from this sample. By default, the spectrum reflects all $\binom{n}{2}$ pairwise comparisons from the full haplotype set $h_0, \ldots, h_{n-1}$ described by `sample.popdata`. To parse all data in this way, use the command

   ```
   python parse_within_pop_allpairs.py sample MaskFolder percentMissing
   ```

   By specifying optional arguments `start1 = i` and `end1 = j`, you can instead sample from restricted haplotype range $h_i, \ldots, h_{j-1}$:

   ```
   python parse_within_pop_allpairs.py sample MaskFolder
           percentMissing start1 end1 start2 end2
   ```

   The `percentMissing` and `maskFolder` arguments work the same here as in `parse_between_pops.py`, described more fully in Section 1.3 on data filtering and quality control.

5. `parse_within_pop_natphase.py`

   This script can be run instead of `parse_within_pop_allpairs.py` with the exact same arguments:

   ```
   python parse_within_pop_natphase.py sample MaskFolder percentMissing
   ```

   or

   ```
   python parse_within_pop_natphase.py sample MaskFolder
           percentMissing start1 end1
   ```

   Instead of generating IBS tracts from all $\binom{n}{2}$ subsampled haplotype pairs, it pairs up haplotype 0 with haplotype 1, haplotype 2 with haplotype 3, and in general, haplotype $2i$ with haplotype $2i+1$. This is designed to sample IBS tracts shared within the input diploid individuals so that the results do not depend on the method that was used to phase the data.

1.3 Data filtering and quality control

Although demography is being inferred from the entire length distribution of IBS tracts, not just long tracts that are correlated with very recent common ancestry, even one or two long IBS tracts may be interpreted as strong evidence for recent migration and/or a population bottleneck. As a result, it is imperative to make sure that the long IBS tracts in your sample are real and are not artifacts of alignment gaps or low/uneven sample coverage. Filtering IBS tracts involves trial and error and is an area in need of more study, but this section describes a couple of basic stategies and tools.

1.3.1 The `percentMissing` argument

We parse haplotype samples such that missing data, annotated as ‘N’s, does not automatically break up IBS tracts. This is designed to keep missing data from degrading real signatures of population size reduction and recent admixture. However, setting the `percentMissing` argument to a threshold t between 0 and 1 will throw out all IBS tracts where the fraction of sites where ‘N’s occur exceeds t. If you rely on this method to control the quality of your IBS tract spectrum and do not specify a folder of mask files, we recommend that you make sure all missing data is annotated in your VCF files. In particular, if every one of your haplotypes is missing data at site s, you should make sure to include s as a site in the VCF file with 'N's across the board.

The ideal threshold t will depend will depend on your sample's sequencing quality and level of polymorphism. We do not specify a default, but recommend that you experiment with different thresholds. The error threshold is probably too low if your longest IBS tracts are clustered in the genome, i.e. if you observe very few tracts shared tracts longer than 200,000 b.p. but see six 200,000 b.p. IBS tracts that occur at the same position in different samples. This sort of pattern is probably the result of missing data or strong positive selection that lead to misleading inferences about demography.

1.3.2 Specifying a mask folder

To exclude specific chromosomal regions from your analysis, you can name a mask folder that includes a file of excluded regions for each chromosome. The package includes a `sampleHumanMaskFolder` that specifies unmappable regions of the human `hg19` reference genome, containing 22 files `chrom1.txt` thru `chrom22.txt`. Note that different maskfiles might be required for other human datasets that have different patterns of mapping errors. You can generate your own mask folder by looking for regions of low mapping quality or SNP sparsity in your own dataset, either excluding a few regions of lowest quality or excluding everything but select high quality regions.

If you want to merge together two mask folders that were created in different ways, you can do so with the script `merge_gapfiles.py`, specifying the names of two existing mask folders in any order followed by the desired name of the output mask folder:

```
python merge_gapfiles.py maskFolder1 maskFolder2 outputFolder
```

For this to work, each mask folder must have the same number of files, one for each chromosome.

2 Optimizing a demographic model

This section describes how to infer population size changes, divergence times, and migration from distributions of IBS tract lengths.

2.1 Setting mutation and recombination rates

Before you start fitting a demographic model to your data, you must hard-code your species's recombination and mutation rates into the file `calc_ibs_backcoal_varmu.py`. This involves modifying the values assigned to `theta` and `rho` at the very beginning of the file, setting `theta` equal to 40,000 times the mutation rate per site per generation and `rho` equal to 40,000 times the recombination rate per site per generation. The factor of 40,000 is motivated by the population genetic formulas $\theta = 4N\mu$ and $\rho = 4Nr$, assuming that the effective population size is on the order of 10,000 as it is in humans.

2.2 Accounting for variable mutation rate

Our coalescent method predicts that the frequency $f_{\text{IBS}}(L)$ of L-base-long IBS tracts is almost constant as a function of L when L is very short (specifically, when $L \ll 1/\Theta$. This prediction holds true in data produced by a coalescent simulator like `ms`. However, in real human data we observe an excess of very short IBS tracts and we attribute this to the presence of cryptic mutation rate variation. As the manuscript explains in more detail, we cope with human mutation rate variation by excluding IBS tracts shorter than 100 base pairs and assuming that a randomly distributed 1% of sites have a mutation rate that is 39 times higher than the baseline mutation rate. These assumptions are encoded by setting the variable `var_mutrate_param` at the start of `demographic_function_builder.py` equal to 0.039 (39 times the mutation rate $\theta = 0.001$ that we use for humans) and setting `min_tract_length` at the start of `infer_from_inputfile.py` equal to 100. If you are working with data from another species, you may want to set these parameters differently, decreasing the minimum tract length used for inference if your organism has a higher mutation rate and setting `var_mutrate_param` equal to `theta` if you do not wish to model cryptic mutational hotspots. To decide whether it is necessary to model cryptic hotspots, you can plot the distribution of observed IBS tract lengths on a log-log scale and check for a sharp uptick in the abundance of the shortest IBS tracts (as seen for human data in Figure 1). The main downside of ignoring IBS tracts shorter than 100 base pairs is the potential for losing information about the ancient past, especially when the mutation rate is high.

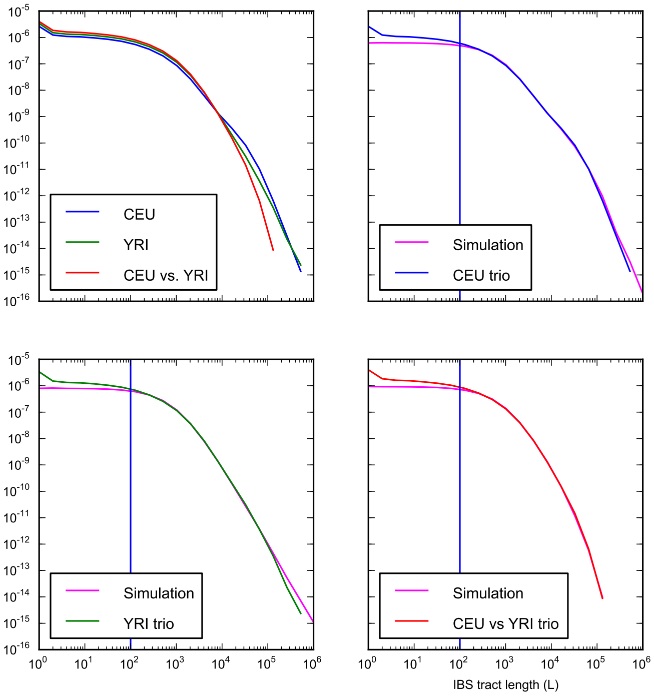

Figure 1: **IBS tracts in real human data vs. simulated data.** These `ms` data shown in pink were simulated under a history fit to the frequency spectrum of human IBS tracts greater than 100 base pairs long. As shown here, the real human data contains an excess of shorter IBS tracts, particularly very short tracts less than 5 base pairs long.

2.3 Inferring size changes within one population

The easiest way to infer size changes within one population is to use the script `infer_onepop_adaptive_cumulative.py` as follows:

```
python infer_onepop_adaptive_cumulative.py population_lengths.ibs
        min_tract_length num_reps generation_time
```

The argument `population.ibs` is the name of the input IBS tract length file. As mentioned earlier, `min_tract_length` is the shortest IBS tract length (measured in base pairs) that you wish to use for inference. It should typically take values between 5 and 100, using shorter values for data with high polymorphism and/or low mutation rate heterogeneity. The third argument, `num_reps`, is the number of iterations you wish to perform: if you set `num_reps` equal to 20, then the script will iteratively find 20 different population histories, each with a better fit and more parameters than the last. Finally, `generation_time` is the desired species generation time in years.

The first history printed to the screen will be the best-fit constant population size history, while the second history will have one population size change and later histories will have additional size changes. At most one population size change will be added each iteration. This is an example of a history that is printed to the screen after being fit to the data:

```
Best history so far:
Population sizes (present to past) :1987.42, 4903.036, 9347.5338
Population size change times (kya) :39.47687, 193.64829

Parameters for plotting:
[0.07895374, 0.30834284, 0.198742, 0.4903036, 0.93475338]
```

These parameters describe a history in which the ancestral effective population size was 9,347. The population size decreased to 4,903 around 194 thousand years ago (kya), and decreased again to 1,987 around 39 kya.

If you wish to plot the expected IBS tracts given this history and compare them to the IBS tracts observed in your data, you need to copy the list of numbers that follows “Parameters for plotting” into a text file called e.g. `demographic_history.txt`. The following command generates a file called demographic_plot.pdf that compares the predicted and observed IBS tract length distributions:

```
python plot_onepop.py population.ibs demographic_history.txt
        demographic_plot.pdf
```

2.4 Inferring a simple population divergence

The script infer_between_adaptive_cumulative.py fits the distribution of IBS tracts shared between two populations to a simple divergence scenario with no migration. It can be called as follows:

```
python infer_onepop_adaptive_cumulative.py pop1_vs_pop2_lengths.ibs
        min_tract_length num_reps generation_time
```

Each output history specifies a divergence time along with a sequence of population size changes that affected the ancestral population prior to divergence:

To check the fit between observed and expected IBS tracts, copy the line following “Parameters for plotting” into a text file called e.g. demographic_history.txt. The following command generates a file called demographic_plot.pdf that compares the predicted and observed IBS tract length distributions:

```
python plot_between.py pop1_vs_pop2.ibs demographic_history.txt
        demographic_plot.pdf
```

If the fit between expected and observed IBS tracts stays poor after many iterations of infer_between_adaptive_cumulative.py, or if the inferred population sizes are not compatible with the sizes inferred by inferred_onepop_adaptive_cumulative.py from the IBS tracts shared within populations, then the data might contain signatures of migration and/or ghost admixture and be incompatible with a simple divergence model.

2.5 Complex population histories

This section describes how to infer the parameters of a complex demographic history by jointly maximizing the likelihood of IBS tracts shared within and between populations. The first step is to specify your demographic model by modifying the file demographic_inputfile.py.

2.5.1 Specifying a demographic model

The file demographic_function_builder.py is not an executable file, but includes a set of functions that you can call to obtain the expected IBS tract length distribution given a custom demographic history. Using these functions you can modify the tract_abundance function in demographic_inputfile.py to specify any history that contains instantaneous population size changes and admixture events, but no continuous periods of migration or population growth.

Here is an example of a tract length distribution function for two populations of sizes N1 and N2 that diverged at time ts from a population of size N and exchanged an admixture pulse of admixture fraction f at time tm. We optimize the time parameter ts_diff = ts - tm rather than the divergence time ts because the divergence time is constrained to be greater than the time of migration, and it is more convenient to simply constrain ts_diff to be greater than zero. The specific functions called here are explained in detail in section 4, but this example is presented first to demonstrate the structure that the distribution function should have:

```
from demographic_function_builder import *
import math

def tract_abundance(L, (uncoalesced_pop1, uncoalesced_pop2,
        uncoalesced_between), (N1, N2, N, tm, ts_diff, f)):

        ts = tm + ts_diff

        prob_list = []
        prob_list = initialize_pop(L, prob_list, uncoalesced_pop1, N1)
        prob_list = initialize_pop(L, prob_list, uncoalesced_pop2, N2)
        uncoalesced_pop1 = time_lapse(uncoalesced_pop1, N1, tm)
        uncoalesced_pop2 = time_lapse(uncoalesced_pop2, N2, tm)
        prob_list, uncoalesced_pop1, uncoalesced_pop2, uncoalesced_between
                = two_way_admixture(L, prob_list, uncoalesced_pop1,
                        uncoalesced_pop2, uncoalesced_between, N1, N2, tm, f, 0)
        uncoalesced_pop1 = time_lapse(uncoalesced_pop1, N1, ts_diff)
        uncoalesced_pop2 = time_lapse(uncoalesced_pop2, N2, ts_diff)
        prob_list, uncoalesced_merged = pop_merge(L,prob_list, uncoalesced_pop1,
                uncoalesced_pop2, uncoalesced_between, N1, N2, N, ts)
```

```
    return math.fsum(prob_list)
```

Here, the object `prob_list` is a list of numbers that will be added together to yield the desired probability. Each demographic event modifies `prob_list` by adding more terms. The python function `math.fsum` adds up the terms of `prob_list` in a way that is more numerically stable than using plus and minus signs.

Demographic events need to be specified in order from the present moving backward into the past. At time zero, the two populations are initialized with the function `initialize_pop` that specifies their current sizes. The scaling of the sizes will depend on the units of your mutation and recombination rate; for example, if you specify `theta = 0.001` mutations per site per $4 \times 10,000$ generations, a typical scaling for humans that corresponds to a mutation rate of 2.5×10^{-8} per site per generation, all population sizes should be scaled by a factor of 10,000. In this case, setting `N1 = 1` and `N2 = 2` means that populations 1 and 2 will have recent sizes 10,000 and 20,000, respectively.

After initializing the populations, the next event backwards in time is the admixture pulse at `tm`, but before that event is specified, the `time_lapse` function must be applied once to each population to move the time pointer back from time `0` to time `tm`. Similarly, after the admixture pulse happens, the `time_lapse` function must be called twice more to move each population's time pointer back to time `ts`.

The three variables `uncoalesced_pop1`, `uncoalesced_pop2`, and `uncoalesced_between` take numerical values between 0 and 1. You can identify them with three functions of time $u_1(t)$, $u_2(t)$, and $u_{12}(t)$, where $u_1(t)$ is the probability that at time t, and randomly selected pair of lineages will not have coalesced yet and will both lie in population 1. Similarly, $u_2(t)$ is the probability that two lineages will not have coalesced yet and both lie in population 1, while $u_{12}(t)$ is the probability that they have not coalesced yet and lie in different populations. The three values that you pass as arguments are $u_1(0)$, $u_2(0)$, and $u_{12}(0)$; if you wish to predict the distribution of IBS tracts shared between two haplotypes sampled from population 1 in the present, you should specify

```
(uncoalesced_pop1, uncoalesced_pop2, uncoalesced_between) = (1,0,0).
```

You can instead specify `(0,1,0)` if both haplotypes are sampled from population 2, or specify `(0,0,1)` if one haplotype is sampled from each population. In summary, the following command returns the frequency of

L-base-long IBS tracts shared between haplotypes from population 1:

```
tract_abundance(L, (1,0,0), (N1, N2, N, tm, ts_diff, f))
```

This command returns the frequence of L-base IBS tracts shared between haplotypes from population 2:

```
tract_abundance(L, (0,1,0), (N1, N2, N, tm, ts_diff, f))
```

This command returns the frequence of L-base IBS tracts shared between one haplotype from population 1 and one haplotype from population 2:

```
tract_abundance(L, (0,0,1), (N1, N2, N, tm, ts_diff, f))
```

In addition to adding terms to `prob_list`, the `time_lapse` function and demographic event functions update `uncoaleced_pop1`, `uncoalesced_pop2`, and `uncoalesced_between` to take on values of $u_1(t)$, $u_2(t)$, and $u_{12}(t)$ for nonzero t. Specifically, the `time_lapse` function decreases the values of `uncoaleced_pop1` and `uncoaleced_pop1` to reflect coalescence events that happen during the elapsed time period. For that reason, you should never apply the `time_lapse` function to `uncoalesced_between` because lineages from different populations cannot coalesce between migration events.

2.6 Specifying parameter bounds

The function `optimization_bounds()` at the start of `demographic_inputfile.py` should always return a list of 2-element lists with the same length as the third argument of `tract_abundance`. For the example history described in 2.5.1, the optimization bounds vector should have length 6.

Setting good parameter bounds can take trial and error. The optimization script will print warnings when parameters are hitting upper and lower bounds, indicating that these bounds need to be raised or possibly lowered (though they should never be set as low as zero). The first three parameters are population size scaling factors defined with respect to the default population size of 10,000; setting each of the first three bounds equal to `[0.05,20.0]` allows the population sizes to vary between 500 and 200,000. The next two arguments are times and should typically be set between 0.001 and 10 or to a narrower range that may make the optimizations take less time and succeed more often. The final parameter is a fraction that should be bounded by `[0.001,0.99]` or some narrower range (setting upper and lower bounds away from 0 and 1 avoids getting the optimization trapped in nested histories that contain no admixture).

2.7 Describing the history

The function describe_history at the end of demographic_inputfile.py allows you to translate the arguments of tract_abudance into more convenient units, for example, converting population size scaling factors into absolute population sizes and converting time difference parameters into absolute times measured in thousands of years before the present (kya). Its only function is to print something to the screen that you will find more useful than the raw optimized parameter vector. It takes the optimized parameter vector as its first argument and the generation time and standard population size as second and third arguments (this function is called from the plotting script plot_from_inputfile.py. Here is an example description of the history from 2.5.1:

```
def describe_history((N1, N2, N, tm, ts_diff, f), gen_time, standard_popsize):
    output = ‘Time of most recent gene flow: ’
            +str(tm*gen_time*standard_popsize*0.001)+‘ kya\n’
    output += ‘Divergence time: ’
            +str((tm+ts_diff)*gen_time*standard_popsize*0.001)+‘ kya\n’
    output += ‘Size of population 1: ’
            +str(N1*standard_popsize)+‘\n’
    output += ‘Size of population 2: ’
            +str(N2*standard_popsize)+‘\n’
    output += ‘Ancestral population size: ’
            +str(N*standard_popsize)+‘\n’
    return output
```

2.8 Running the optimization

Once you have specified a history by customizing demographic_inputfile.py, you can optimize its parameters as follows:

```
python optimize_from_inputfile.py pop1.ibs pop2.ibs
        pop1_vs_pop2.ibs optimization_output.txt min_tract_length
```

In this command line, optimization_output.txt is the desired name of the optimization output file. The parameter min_tract_length is the length in base pairs of the shortest IBS tracts to be included in the optimization (as discussed before, this should typically be set between 5 and 100). Warnings will pop up if you are hitting the upper and/or lower bounds of the optimization. The parameters are numbered starting at 0–if, for example, parameter 3 hits its upper optimization bound, you should manually increase the second number of the fourth entry of the vector returned by `optimization_bounds` in `demographic_inputfile.py`:

```
def optimization_bounds():
return [[0.001,0.05],[0.001,0.1],[0.0001,0.3],[0.001,1],[0.05,10]]
```

changes to

```
def optimization_bounds():
return [[0.001,0.05],[0.001,0.1],[0.0001,0.3],[0.001,3],[0.05,10]]
```

There is less of a problem if parameters are hitting lower bounds; this only indicates that the model could be simplified to eliminate some parameters.

2.9 Plotting expected vs. observed IBS tract lengths

Once you have optimized the parameters of your custom history, you can plot the expected vs. observed IBS tracts using the following command:

```
python plot_from_inputfile.py pop1.ibs pop2.ibs
        pop1_vs_pop2.ibs optimization_output.txt desired_graph_filename.pdf
        generation_time min_tract_length
```

Your description of the demographic history being plotted will print to the screen. The inferred and observed IBS tract length distributions will be plotted in a pdf with the name `desired_graph_filename.pdf`. If the model does not appear to fit the data well, it should not be considered a reliable demographic estimate. To improve the fit, you may need to change the optimization bounds, try a more complex model, adjust the variable mutation rate parameter, or remove more missing data-heavy long IBS tracts from the observed IBS tract length spectrum.

3 Simulating IBS tract lengths with `ms`

To assess the accuracy of an inferred history and test for parameter estimate bias, it is good to perform parametric bootstrapping, simulating several replicate datasets under the full coalescent with recombination and estimating demographic parameters from each dataset. The script `ibs_from_ms_output.py` can parse an output file `ms.out` of Hudson's `ms` coalescent simulator [1] and return the IBS tract length distribution of the data:

```
python ibs_from_ms_output.py ms.out rep_sequence_length simulated_tracts.ibs
```

The second argument, `rep_sequence_length`, is the length in base pairs of each "chromosome" being simulated. The third argument, `simulated_tracts.ibs`, is the desired name of the IBS tract length file.

For simulating long chromosomes, it is necessary to increase the decimal precision of the `ms` segregating site position output. To do this, find the line `fprintf(pf, "%6.41f  ",posit[i] );` in the source code `ms.c`, replace it with `fprintf(pf, "%6.10lf ",posit[i]);`, and recompile `ms` with the command `gcc -o ms ms.c streec.c rand1t.c -lm`.

4 List of demographic event functions

This section explains all of the functions that can be imported from `demographic_function_builder.py`. Most of these functions are related to commands from Hudson's coalescent simulator `ms` [1], and we include these commands for the benefit of users who are already familiar with `ms` syntax.

```
prob_list = initialize_pop(L,prob_list,uncoalesced_pop_i, N)
```

- Sets the initial size of population i equal to `N`.
- Corresponding `ms` command: `-en 0 i N`

```
uncoalesced_pop_i = time_lapse(uncoalesced_pop_i, N, t-t_0)
```

- Advances the time pointer in population i from time `t_0` to time `t`

- Before calling this function, ensure that population i's time pointer is currently at time `t_0` and that its population size is currently `N`
- Corresponding `ms` command: N/A

```
prob_list = popsize_change(L,prob_list,uncoalesced_pop_i, N_old, N, t)
```

- At time `t`, changes the size of population i from `N_old` to `N`.
- Before calling this function, check that the population i has current size `N_old` and its time pointer has been advanced to time `t` using `time_lapse`.
- Corresponding `ms` command: `-en t i N`

```
prob_list, uncoalesced_pop1, uncoalesced_pop2, uncoalesced_merged
          = two_way_admixture(L,prob_list,uncoalesced_pop1,
         uncoalesced_pop2, uncoalesced_between, N1, N2, t, f1, f2)
```

- At time `t`, looking backward in time, each lineage from population 1 moves to population 2 with probability `f1`. Each lineage from population 2 moves to population 1 with probability `f2`.
- Before calling this function, check that the populations 1 and 2 have current sizes `N1` and `N2`, respectively, and that both time pointers have been advanced to time `t` using `time_lapse`.
- Corresponding `ms` command:

  ```
  -es t 1 1-f1 -es 2 1 1-f2 -ej t 3 2 -ej t 4 1
  ```

```
prob_list, uncoalesced_pop1, uncoalesced_pop2, uncoalesced_merged
         = two_way_admixture_change_one_size(L,prob_list,uncoalesced_pop1,
         uncoalesced_pop2, uncoalesced_between, N1_old, N1, N2, t, f1, f2)
```

- Same as `two_way_admixture` except that population 1 changes size from `N1_old` to `N1` at the same time as the admixture event. This is more numerically stable than specifying separate admixture and size change events.

- Before calling this function, check that the populations 1 and 2 have current sizes N1_old and N2, respectively, and that both time pointers have been advanced to time t using time_lapse.
- Corresponding ms command:

  ```
  -es t 1 1-f1 -es 2 1 1-f2 -ej t 3 2 -ej t 4 1 -en t 1 N1
  ```

```
prob_list, uncoalesced_pop1, uncoalesced_pop2, uncoalesced_merged
        = two_way_admixture_change_one_size(L,prob_list,uncoalesced_pop1,
        uncoalesced_pop2, uncoalesced_between, N1_old, N2_old, N1, N2, t, f1, f2)
```

- Same as two_way_admixture except that both populations change size at the same time as the admixture event. This is more numerically stable than specifying separate admixture and size change events.
- Before calling this function, check that the populations 1 and 2 have current sizes N1_old and N2_old, respectively, and that both time pointers have been advanced to time t using time_lapse.
- Corresponding ms command:

  ```
  -es t 1 1-f1 -es 2 1 1-f2 -ej t 3 2 -ej t 4 1 -en t 1 N1 -en 2 N2
  ```

```
prob_list, uncoalesced_merged = pop_merge(L,prob_list,uncoalesced_pop1,
        uncoalesced_pop2, uncoalesced_between, N1_old, N2_old, N, t)
```

- Corresponds to population splitting forward in time. At time t looking backward in time, two populations of sizes N1 and N2 merge into a single population of size N.
- Before calling this function, check that the populations 1 and 2 have current sizes N1 and N2, respectively, and that both time pointers have been advanced to time t using time_lapse.
- Corresponding ms command: `-ej t 2 1 -en t 1 N`

```
prob_list, uncoalesced_main, uncoalesced_ghost, uncoalesced_between
= ghost_pop_split(L,prob_list,uncoalesced_pop_i, N_old, N_main, N_ghost, t, f)
```

- Corresponds to ghost admixture forward in time. At time `t` looking backward in time, a population of size `N_old` transfers a fraction `f` of its lineages into a new ghost population of size `N_ghost`. The remaining lineages contribute to a main population of size `N_main`.
- Before calling this function, check that the population *i* has current size `N_old` and that its time pointer has been advanced to time `t` using `time_lapse`.
- Corresponding `ms` command: `-es t 1 1-f -en t 1 N_main -en t 2 N_ghost`

4.1 Joint histories of three or more populations

A little extra bookkeeping is required to compute the expected IBS tract length distribution for a joint history of three or more populations. This applies even if DNA is sampled from only two extant populations but is a third ghost population splits off at some point in the past. The extra thing to remember is that when populations 1 and 2 split apart, merge, or exchange migrants, you must manually keep track of pairs of lineages that are split between populations 1 and 3 or between populations 2 and 3.

To illustrate, here is an example of a tract length distribution for a history including ghost admixture:

```
from demographic_function_builder import *
import math

def tract_abundance(L, (uncoalesced_pop1, uncoalesced_pop2,
        uncoalesced_pop1_pop2), (N, t_ghost_mix, t_split_diff,
        t_ghost_split_diff, f)):

        t_split = t_ghost_mix+ t_split_diff
        t_ghost_split = t_split + t_ghost_split_diff

        prob_list = []
        prob_list = initialize_pop(L,prob_list, uncoalesced_pop1, N)
        prob_list = initialize_pop(L,prob_list, uncoalesced_pop2, N)
        uncoalesced_pop1 = time_lapse(uncoalesced_pop1, N, t_ghost_mix)
        uncoalesced_pop2 = time_lapse(uncoalesced_pop2, N, t_ghost_mix)
        prob_list, uncoalesced_pop1, uncoalesced_ghost,
                uncoalesced_pop1_ghost = ghost_pop_split(L, prob_list,
                uncoalesced_pop1, N, N, N, t_ghost_mix, f)
        uncoalesced_pop2_ghost = uncoalesced_pop1_pop2*f   # extra bookkeeping
        uncoalesced_pop1_pop2 = uncoalesced_pop1_pop2*(1-f)   # extra bookkeeping
        uncoalesced_pop1 = time_lapse(uncoalesced_pop1, N, t_split_diff)
        uncoalesced_pop2 = time_lapse(uncoalesced_pop2, N, t_split_diff)
        uncoalesced_ghost = time_lapse(uncoalesced_ghost, N, t_split_diff)
        prob_list, uncoalesced_main = pop_merge(L, prob_list,
                uncoalesced_pop1, uncoalesced_pop2, uncoalesced_pop1_pop2,
                        N, N, N, t_split)
        uncoalesced_main_ghost = uncoalesced_pop1_ghost
                + uncoalesced_pop2_ghost        #extra bookkeeping
        uncoalesced_main = time_lapse(uncoalesced_main, N, t_ghost_split_diff)
        uncoalesced_ghost = time_lapse(uncoalesced_ghost, N, t_ghost_split_diff)
        prob_list, uncoalesced_merged = pop_merge(L, prob_list,
                uncoalesced_main, uncoalesced_ghost, uncoalesced_main_ghost,
                N, N, N, t_ghost_split)

        return math.fsum(prob_list)
```

References

[1] Hudson R (2002) Generating samples under a Wright-Fisher neutral model of genetic variation. Bioinformatics 18: 337–338.