IBDcreate Documentation

October 30, 2013

Authors

Programs simu, ibd_states, create, and beaglesim written by Chris Glazner, together with examples written/compiled by Marshall Brown, Also included are the beaglesim R-code and test examples.

The C/R routines were written by Chris Glazner; 2010 The original version of this README was created on 4/15/2011 by Marshall Brown

Resorting of the programs and functions, and rewriting of README was by Elizabeth Thompson 7/22/2011

This version, including renaming of programs and update of README is by Fiona Grimson.

Contents

1 Simulation of haplotypes

1.1 beaglesim

Summary: beaglesim is a function for the simulation of haplotypes at varying levels of LD, based upon a BEAGLE model.

1. Make the executable by using `make beaglesim` in the main directory.
2. Run BEAGLE on your haplotype data to create the model .dag.dz file.
3. Put the name of the .dag.gz file into beagle_config.txt in the Gold subdirectory, and edit any other items of config.txt as desired (for details, see the comments in that file)
4. Run in the Gold subdirectory:

   ```
   ../beaglesim beagle_config.txt
   ```

   an output file "myhaps.txt" will be created.

2 Simulation of IBD

The C programs in this section deal with the simulation of identity by descent (IBD).

2.1 simpop_fgl

Summary: Simulates patterns of identity by descent in a population after a user-specified number of generations. simpop_fgl assigns founder genome labels (FGL) to the founder generation, and simulates new generations under a 2-sex diploid population model that approximates random mating. The program uses a user-specified avg. number of recombinations/Morgan.

Compared to the earlier version (simu.c), simpop_fgl also has the option of specifying a remating fraction to reduce the probability of multiple marriages per individual, thereby reducing the number of half-siblings in the population.

For each simulated chromosome, identification information is printed (to stdout) along with chromosome junction data in [FGL endpoint] format.

Usage: in the Gold subdirectory

```
../simpop_fgl [generation size/2][# generations][avg. # recombs/Morgan]
              [# of generations to print][chrom. length in base pairs]
              [remating fraction][(optional) seed]
```

The inputs are

- `generation size/2`: i.e. number of females in each generation
- `# generations`: number generations after founders
- `# recombs/Morgan`: average number of recombinations per Morgan

- `# of generations to print`: prints at the end, starting from the last generation
- `chrom. length in base pairs`
- `remating fraction`: This input is a parameter affecting the selection of parents for each mating. If the remating fraction is $r \in (0, 1]$, the probability of a randomly sampled individual being accepted as a parent in a mating is r^m, where m is the number of previous matings of that individual. If the individual is not accepted, another is sampled at random and the process repeats until a parent is accepted.
- `(optional) seed`: set from system clock if omitted

The output prints one line to stdout for each chromosome in the following format:

```
[person id #] [generation #] [mother id #] [father id #]
    [sex (0=female)] [which chromsome (0=maternal)]
```

followed by the chromosome junction data in the form:

```
[fgl label] [endpoint] [fgl label] [endpoint] ...
```

Note that the females are printed first, then the males, before progressing to the next generation.

Example: In the Gold subdirectory

```
../simpop_fgl 1 4 1.0 5 100000000 1 7654
```

This is a simulated population that began with one male and one female (four founder genome labels) and produced 5 generations, each consisting of the two offspring of the two individuals in the previous generation. My computer produces:

```
1 0 0 0 0 0 1 100000000
1 0 0 0 0 1 2 100000000
2 0 0 0 1 0 3 100000000
2 0 0 0 1 1 4 100000000
3 1 1 2 0 0 1 67366937 2 100000000
3 1 1 2 0 1 4 98753700 3 100000000
4 1 1 2 1 0 2 100000000
4 1 1 2 1 1 4 100000000
5 2 3 4 0 0 4 98753700 3 100000000
5 2 3 4 0 1 2 100000000
6 2 3 4 1 0 1 66920241 4 98753700 3 100000000
6 2 3 4 1 1 4 14424740 2 100000000
7 3 5 6 0 0 2 100000000
7 3 5 6 0 1 4 14424740 2 100000000
8 3 5 6 1 0 4 94850878 2 100000000
8 3 5 6 1 1 1 13587054 4 14424740 2 100000000
```

```
9 4 7 8 0 0 2 100000000
9 4 7 8 0 1 4 38591150 2 100000000
10 4 7 8 1 0 2 100000000
10 4 7 8 1 1 1 13587054 4 14424740 2 39641642 4 81801489 2 100000000
```

For example, if we look at the fifth line of output we see that on the first chromosome of individual 3, there is a recombination between bases 67366937 and 67366938, with the fgl 1 being allocated to the [1,67366937] segment of chromosome, and fgl 2 over [67366938, 100000000].

If the seed is not specified, it is set from the system clock and output will differ from run to run.

The remating fraction here is set to 1, but may be any value in (0,1] and will default to 1 if a larger value is supplied.

This example output is given in the file sim_Gold in the Gold subdirectory. An example of 100 individuals from a larger simulation is in the file fglhaps_first100.txt

2.2 fgl2haplo

Summary: Assigns haplotype segments to the output of simpop_fgl to create chromosomes with simulated patterns of coancestry.

This function was derived from create.c, without the calculation of patterns of genotypic and haplotypic IBD states which can now be found in fgl2ibd. Some improvements were also made to memory usage, as well as the addition of the recycling option.

Usage: In the Gold subdirectory:

```
../fgl2haplo [simulated_fgl_file] [founderhap_file] [SNPs_metadata_file]
             [Which_to_create] [Output with spaces and labels?]
```

The inputs are:

- `simulated_fgl_file`: output from simpop_fgl
- `founderhap_file`: Haplotype data – one row for each marker, no spaces between alleles. These data can be created by beaglesim. If the program requires more haplotypes than are provided to create this many individuals, an error will be printed.
- `SNPs_metadata_file`: contains the following data, each on a new line.

  ```
  [ number of haplotypes to input]
  [ number of SNPs]
  [ marker pos 1 in bp]
  [ marker pos 2 in bp]
  ...
  ```

- `Which_to_create`: either an integer or a file name.

If it is an integer value, say n, the first n individuals from simulated_fgl_file will be created.

If a file is named, the file should specify which individuals are required from simulated_fgl_file in the order in which they should be printed. Each index appears on a new line. For example, if the third, fifth, then fourth individuals are required, the file would read

```
3
5
4
```

Note that the index of the individual indicating their position in simulated_fgl_file should be supplied, not their ID number.

- `Output with spaces and labels?`: should the id number of the individual be printed at the start of each of their haplotypes, and spaces placed between each allele?

Example: In the Gold subdirectory

```
../fgl2haplo fglhaps_first100.txt rawalleles_broken_Gold.txt
        metadata_Gold.txt 10 1> haplo_examp.out
```

which produces the (stdout) output file `haplo_examp.out`. A "Gold" version of this files is given in the Gold subdirectory.

Recycling Example: The ability to supply the `Which_to_create` file with a shuffled list of haplotypes to print was to allow the "recycling" of a simulated pedigree. The idea is to

1. Create founder haplotypes using beaglesim
   e.g. 1917 founder haplotypes in `myhaps.txt`
2. Create m generation pedigree using simpop_fgl and save the final generation.
   e.g. A 10-person 10-generation pedigree

   ```
   ../simpop_fgl 5 10 1 1 1000000000 1 1234 > simfgl_10.txt
   ```

3. Combine founder haplotypes and pedigree to create mth generation halotypes. Shuffle the order of the mth generation haplotypes.
   e.g. Using `randomOrder1.txt` as the order in which to output the 10 individuals of generation 10. Note that we do not print with spaces and labels if the output is to be used again in fgl2haplo as it must look like beaglesim output.

   ```
   ../fgl2haplo simfgl_10.txt myhaps.txt metadata_Gold.txt
         randomOrder1.txt 0 > myhapsRecycle10.txt
   ```

4. Combine the shuffled mth generation haplotpes with the pedigree to create the $2m$th generation hapolotypes
   e.g. Now use `myhapsRecycle10.txt` as the "founders" and `randomOrder2.txt` as output ordering for generation 20.

```
../fgl2haplo simfgl_10.txt myhapsRecycle10.txt metadata_Gold.txt
      randomOrder2.txt 0 > myhapsRecycle20.txt
```

5. etc

2.3 fgl2ibd

Summary: This program produces files that contain the pairwise patterns of genotypic (nine) and haplotypic (fifteen) IBD states among the created individuals at the given marker positions.

Usage:

```
../fgl2ibd [simulated_fgl_file] [SNPs_metadata_file] [# of people to create]
```

The inputs are:

- `simulated_fgl_file`: output from simpop_fgl
- `SNPs_metadata_file`: contains the following data, each on a new line.

  ```
  [ number of haplotypes to input]
  [ number of SNPs]
  [ marker pos 1 in bp]
  [ marker pos 2 in bp]
  ...
  ```

- `# of people to create`: This should be an integer, say n, the first n individuals from simulated_fgl_file will be created.

Example: In the Gold subdirectory:

```
../fgl2ibd fglhaps_first100.txt metadata_Gold.txt 10
```

The outputs are:

- outfifteenibd.txt : pairwise IBD for haplotypic data
- outnineibd.txt: pairwise IBD for genotypic data

"Gold" versions of these files are found in the Gold subdirectory.

3 Makefile

The Makefile provides the make commands for the C programs: beaglesim, simpop_fgl, fgl2haplo and fgl2ibd. The following tasks can be performed using the Makefile:

- `make <progname>` Make each program
- `make all` Make all four programs
- `make clean_progs` Remove the 4 executables

- `make clean_beagle` Removes the files created by beaglesim
- `make clean_fgl2haplo` Removes the files created br fgl2haplo
- `make clean_fgl2ibd` Removes the files created br fgl2ibd
- `make clean_data` Removes all these data files
- `make ultraclean` Remove both executables and created data files.

4 Gold Subdirectory contents

The Gold subdirectory contains the following files:

- `beagle_phased.dag.gz` beaglesim input: the BEAGLE model
- `beagle_config.txt` beaglesim config file
- `myhaps.txt` beaglesim output
- `sim_Gold.txt` small example output from simpop_fgl
- `fglhaps_first100.txt` simpop_fgl output used in fgl2haplo and fgl2ibd
- `rawalleles_broken_Gold.txt` beaglesim output haplotypes used by fgl2haplo
- `metadata_Gold.txt` marker positions; used by fgl2haplo and fgl2ibd
- `haplo_examp_Gold.out` output haplotype data from fgl2haplo
- `outnineibd_Gold.txt` ibdstate output from fgl2ibd
- `outfifteenibd_Gold.txt` ibdstate output from fgl2ibd
- `randomOrder1.txt` random ordering used in Recycling example
- `randomOrder2.txt` random ordering used in Recycling example
- `simfgl_10.txt` simulated pedigree from Recycling example
- `myhapsRecycle10.txt` output from Recycling example
- `myhapsRecycle20.txt` output from Recycling example