# MORGAN V3.4 Tutorial

Version 3.4
Last Updated: 6 September 2019

# Table of Contents

**1 Get Started . . . 1**

1.1 Overview of MORGAN . . . 1
1.2 Get the Tutorial . . . 3
1.3 Get and set up the examples . . . 3
1.4 Overview of the pedigrees used in the examples . . . 5
1.5 Structure of the MORGAN package . . . 5

**2 Common Features and File Formats . . . 9**

2.1 Command syntax . . . 9
2.2 Parameter file . . . 10
2.3 File identification statements . . . 11
2.4 Limit overriding . . . 12
2.5 Input extra variables . . . 13
2.6 Output control . . . 13
2.7 Pedigree file . . . 14
2.8 Individuals file . . . 15
2.9 Pedigree file description statements . . . 15

**3 Checking Pedigree Validity . . . 18**

3.1 Introduction to `pedcheck` . . . 18
3.2 Sample `pedcheck` parameter file . . . 18
3.3 Running `pedcheck` examples . . . 19
3.4 Creating an `individuals file` . . . 20
3.5 `pedcheck` statements . . . 20

**4 Computing Kinship and Other Pedigree Computations . . . 22**

4.1 Introduction to `kin` . . . 22
4.2 Sample `kin` parameter file . . . 22
4.3 Running `kin` example and sample output . . . 23
4.4 `kin` statements . . . 24
4.5 The `translink` utility . . . 25
4.6 The `ibd_class` utility . . . 26

**5 Simulating Marker and Trait Data in Pedigrees . . 29**

5.1 Introduction to `genedrop` . . . 29
5.2 Sample `genedrop` parameter file . . . 30
5.3 Running `genedrop` examples and sample output . . . 32
5.4 `genedrop` statements . . . 33
5.4.1 `genedrop` computing requests . . . 33
5.4.2 `genedrop` mapping model parameters . . . 33

5.4.3 genedrop population model parameters ... 34
5.4.4 genedrop computational parameters ... 35
5.4.5 genedrop output pedigree options ... 35
5.4.6 genedrop output seed file options ... 36

**6 Simulating Marker Data Conditional on Trait Data in Pedigrees ... 37**

6.1 Introduction to markerdrop ... 37
6.2 Sample markerdrop parameter file – conditional on trait ... 37
6.3 Sample markerdrop parameter file – conditional on inheritance pattern ... 40
6.4 Running markerdrop examples and sample output ... 41
6.5 markerdrop statements ... 43
6.5.1 markerdrop computing requests ... 43
6.5.2 markerdrop mapping model parameters ... 44
6.5.3 markerdrop population model parameters ... 44
6.5.4 markerdrop computational parameters ... 45
6.5.5 markerdrop input file options ... 45

**7 Estimating *a priori ibd* Probabilities by Monte Carlo ... 46**

7.1 Introduction to ibddrop ... 46
7.2 Sample ibddrop parameter file ... 47
7.2.1 Classic ibddrop parameter file ... 47
7.2.2 Using dense marker simulation ... 48
7.2.3 Selection against autozygosity ... 50
7.3 Running ibddrop example and sample output ... 51
7.4 ibddrop statements ... 54

**8 Using MCMC to Estimate Parameters of Interest in Pedigree Data ... 56**

8.1 Specifying inheritance ... 56
8.2 Genetic model ... 56
8.3 Exact HMM computations ... 57
8.4 Single and multiple meiosis LM-samplers ... 58
8.5 MCMC computational options ... 59
8.6 MCMC parameter statements ... 61

**9 Estimating Conditional *ibd* Probabilities by MCMC . . . . . . . . . . . . . . . . . . . . . . . 63**

9.1 Introduction to `lm_auto`, `gl_auto` and `lm_pval` . . . . . . . . . . . . . . . 63
9.2 Sample `lm_auto` parameter file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
9.3 Running `lm_auto` example and sample output . . . . . . . . . . . . . . . . 67
9.4 Sample `gl_auto` parameter file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
9.5 Running `gl_auto` example and sample output . . . . . . . . . . . . . . . . 71
9.6 Sample `lm_pval` parameter file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
9.7 Running `lm_pval` example and sample output . . . . . . . . . . . . . . . . 73
9.8 Autozyg statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
9.8.1 Autozyg computing requests . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
9.8.2 Autozyg file identification statements . . . . . . . . . . . . . . . . . . . . 75
9.8.3 Autozyg pedigree file description . . . . . . . . . . . . . . . . . . . . . . . . . 76
9.8.4 Autozyg output file description . . . . . . . . . . . . . . . . . . . . . . . . . . 76
9.8.5 Autozyg mapping model parameters . . . . . . . . . . . . . . . . . . . . . . 76
9.8.6 Autozyg population model parameters . . . . . . . . . . . . . . . . . . . . 77
9.8.7 Autozyg computational parameters . . . . . . . . . . . . . . . . . . . . . . . 77
9.8.8 Autozyg MCMC parameters and options . . . . . . . . . . . . . . . . . . 79

**10 Estimating *ibd* Based Test Statistics by MCMC . . . . . . . . . . . . . . . . . . . . . . . . . . . 80**

10.1 Introduction to `lm_ibdtests` and `civil` . . . . . . . . . . . . . . . . . . . . . 80
10.2 Sample `lm_ibdtests` parameter file . . . . . . . . . . . . . . . . . . . . . . . . . . 81
10.3 Sample `lm_ibdtests` output . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
10.4 Sample `civil` parameter file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
10.5 Sample `civil` output . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
10.6 `lm_ibdtests` and `civil` statements . . . . . . . . . . . . . . . . . . . . . . . . . 87

**11 Estimating Location lod Scores by MCMC . . 89**

11.1 Introduction to `lm_linkage`, `lm_bayes`, `lm_twoqtl`, `gl_lods` and `base_trait_lods` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89
11.2 Sample parameter files for `lm_linkage` and `lm_bayes` . . . . . . . . . 91
11.3 Running `lm_linkage` examples and sample output . . . . . . . . . . . . 95
11.4 Running `lm_bayes` examples and sample output . . . . . . . . . . . . . . 96
11.5 Running `lm_twoqtl` examples and sample output . . . . . . . . . . . . . 97
11.6 Parameter files for the `gl_lods` program . . . . . . . . . . . . . . . . . . . . 101
11.7 Running `gl_lods` examples and sample output . . . . . . . . . . . . . . 105
11.8 Running the `base_trait_lods` program . . . . . . . . . . . . . . . . . . . . 109
11.9 Location lod scores statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111
11.9.1 Location lod scores computing requests . . . . . . . . . . . . . . . . . 111
11.9.2 Location lod scores file identification statements . . . . . . . . 111
11.9.3 Location lod scores pedigree file description . . . . . . . . . . . . . 112
11.9.4 Location lod scores output file description . . . . . . . . . . . . . . 112
11.9.5 Location lod scores mapping model parameters . . . . . . . . . 112
11.9.6 Location lod scores population model parameters . . . . . . . 113
11.9.7 Location lod scores computational parameters . . . . . . . . . . 113
11.9.8 Location lod scores MCMC parameters and options . . . . . 115

**12 Population-based inference of IBD . . . . . . . . . 117**

12.1 Introduction to ibd_create and ibd_haplo . . . . . . . . . . . . . . . . . . . . 117
12.2 Sample parameter files for ibd_create; `beaglesim` and `beagledag` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120
12.3 Running ibd_create examples and sample output; beaglesim and beagledag . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
12.4 Sample parameter files for ibd_create; simpop_fgl . . . . . . . . . . . . 122
12.5 Running ibd_create examples and sample output; simpop_fgl . . 123
12.6 Sample parameter files for ibd_create; fgl2ibd and fgl2haplo and fgl2dgl . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123
12.7 Running ibd_create examples and sample output; fgl2ibd and fgl2haplo and fgl2dgl . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126
12.8 Sample parameter files for ibd_haplo . . . . . . . . . . . . . . . . . . . . . . . . . 127
12.9 Running ibd_haplo examples and sample output . . . . . . . . . . . . . 129
12.10 Population-based IBD inference parameter statements . . . . . . 130
12.10.1 beaglesim and beagledag parameter statements . . . . . . . . 130
12.10.2 simpop_fgl parameter statements . . . . . . . . . . . . . . . . . . . . . . 131
12.10.3 fgl2ibd and fgl2haplo and fgl2dgl parameter statements . . 132
12.10.4 ibd_haplo parameter statements . . . . . . . . . . . . . . . . . . . . . . . 133

**13 Polygenic Modeling of Quantitative Traits by EM Algorithm . . . . . . . . . . . . . . . . . . . . . . 135**

13.1 Introduction to PolyEM programs . . . . . . . . . . . . . . . . . . . . . . . . . . 135
13.2 Sample `multivar` parameter file . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135
13.3 Running multivar example and sample output . . . . . . . . . . . . . . . 137
13.4 `multivar` statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
13.4.1 `multivar` computing requests . . . . . . . . . . . . . . . . . . . . . . . . . . 139
13.4.2 `multivar` segregation model parameters . . . . . . . . . . . . . . . . 139
13.4.3 `multivar` computational parameters . . . . . . . . . . . . . . . . . . . . 140
13.4.4 `multivar` computational options . . . . . . . . . . . . . . . . . . . . . . . . 141
13.4.5 `multivar` output options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 141

**14 Estimating Genetic Maps from Marker Data . . 143**

14.1 Introduction to `lm_map` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 143
14.2 Sample `lm_map` parameter file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 143
14.3 Running `lm_map` with genotypic data . . . . . . . . . . . . . . . . . . . . . . . 146
14.4 Running `lm_map` with phenotypic data . . . . . . . . . . . . . . . . . . . . . . 146
14.5 `lm_map` statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147

**References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 149**

**Concept Index . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 151**

**Statement Index . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155**

# 1 Get Started

## 1.1 Overview of MORGAN

MORGAN (Monte Carlo Genetic Analysis) is a collection of programs and libraries developed at the University of Washington under the PANGAEA (`http://www.stat.washington.edu/thompson/Genepi/pangaea.shtml`) (Pedigree Analysis for Genetics and Epidemiological Attributes) umbrella. This software implements a number of methods for the analysis of data observed on members of a pedigree, with the main programs implementing Markov Chain Monte Carlo (MCMC) methods. As of the date of this tutorial, the latest MORGAN version is 3.4 which was released in August 2016. It is available for download through the MORGAN (`http://www.stat.washington.edu/thompson/Genepi/MORGAN/Morgan.shtml`) home page at the Department of Statistics, University of Washington.

The MORGAN programs are grouped into five categories:

1. Programs using deterministic algorithms: `pedcheck` checks for errors in pedigree structure and data format, see Chapter 3 [Checking Pedigree Validity], page 18. `kin` computes kinship and inbreeding coefficients for members of the pedigree, see Chapter 4 [Computing Kinship and Other Pedigree Computations], page 22. Additionally, the utility programs `translink` and `ibd_class` are also included in this group of programs.
2. Programs using simple Monte Carlo techniques (by simulating data on founders and '`dropping`' genes down the pedigree): `genedrop` simulates data on a pedigree for analysis by other programs, see Chapter 5 [Simulating Marker and Trait Data in Pedigrees], page 29. `markerdrop` simulates marker data at loci linked to a trait locus, see Chapter 6 [Simulating Marker Data Conditional on Trait Data in Pedigrees], page 37. `ibddrop` uses Monte Carlo to estimate gene *ibd* (identity by descent) probabilities in the absence of data, see Chapter 7 [Estimating a priori IBD Probabilities by Monte Carlo], page 46.
3. Programs using Markov chain Monte Carlo (MCMC) techniques: MORGAN's MCMC programs are split into two sections, `Autozyg` and `Lodscore`. These programs typically have the prefix '`lm_`' indicating that they use the MCMC LM-sampler. In fact, most of these programs now have the option to use the improved multiple-meiosis LM-sampler. A brief introduction to the MCMC sampling techniques employed by MORGAN can be found in Chapter 8 [Using MCMC to Estimate Parameters of Interest in Pedigree Data], page 56.

   The `Autozyg` and `Lodscore` programs may be categorized into five subsets:

   - The Autozyg programs `lm_auto`, `gl_auto` and `lm_pval`, estimate conditional gene *ibd* probabilities; see Chapter 9 [Estimating Conditional IBD Probabilities by MCMC], page 63.
   - The Autozyg programs `lm_ibdtests` and `civil` realize inheritance conditional on genetic marker data and uses these realizations to estimate *ibd*-based test statistics for linkage detection; see Chapter 10 [Estimating IBD Based Test Statistics by MCMC], page 80.
   - The Lodscore programs `lm_linkage`, `lm_bayes` and `lm_twoqtl`, estimate multipoint lod scores; see Chapter 11 [Estimating Location lod Scores by MCMC], page 89.

- The Lodscore program `gl_lods` is analogous to `lm_linkage` but uses IBD generated by `gl_auto` to compute lod scores directly from IBD generated conditional on marker data. The `base trait_lods` program may be used to compute a normalizing base log-likelihood for the log-likelihoods produced by `gl_lods`. Added for MORGAN V3.4, the gl_lods program now provides the 5th and 95th quantiles of the lod score realizations, rather than attempting a Monte V=Carlo standard error computation.
- The Autozyg program `lm_map` realizes inheritance conditional on genetic marker data, and uses these realizations in the estimation of genetic maps; see Chapter 14 [Estimating Genetic Maps from Marker Data], page 143.

4. Programs that use population-based models rather than pedigree information; see Chapter 12 [Population-based inference of IBD], page 117. These programs are in the IBD_Haplo program directory.
   - The IBD_Haplo `ibd_create` program consists of seven subprograms. These simulate population or pedigree descent of IBD, generate or analyze haplotypes in a population, and convert IBD to other forms of IBD specification, or haplotypic information. For details, see Chapter 12 [Population-based inference of IBD], page 117.
   - The IBD_Haplo program `ibd_haplo` produces estimates of IBD given genetic marker data, but uses a population model rather than pedigree information.
5. Programs using EM algorithm for segregation analysis with quantitative traits: includes `univar`, `unibig`, `bivar` and `multivar`, see Chapter 13 [Polygenic Modeling of Quantitative Traits by EM Algorithm], page 135.

This tutorial is based on the tutorial and examples for MORGAN 2.9 developed over the years 2002-2010 by Elizabeth Thompson, Michael Na Li, Myrna Jewett, Adele Mitchell, Audrey Fu, Tia Lerud, and Marshall Brown. MORGAN 2.9 and its accompanying tutorial remain available for download. Adam Gustafson updated the examples and tutorial for the new parameter statements of MORGAN 3.0 in 2011, and the text has subsequently been significantly updated and revised by Elizabeth Thompson.

The version for MORGAN 3.1.1 (December 2012), contains a new Chapter and examples for the new `ibd_haplo` program, while the updated version for MORGAN 3.2 (January 2014) contains a variety of updates, mainly associated with `ibd_haplo` and `gl_lods`. The version for MORGAN 3.3 of May 2015 greatly expands the Chapter on population-based *ibd*, with text and examples for the seven subprograms of `ibd_create`. Additionally the examples for `ibd_haplo` and `gl_lods` have been updated, and the `MORGAN_Examples` directory has been expanded to include these new examples for all three programs. A number of other small edits and updates were made. For version 3.3.2 (August 2016) there are additional changes and improvements to the `ibd_create` subprograms, including the addition of two new subprograms `simped_fgl` and `fgl2dgl`. A new program `base_trait_lods` has also been added. This program provides the probability of trait data on a pedigree. This probability can be used to normalize the ibd-based lod score computed by `gl_lods`. Some other minor changes to parameter statements (for example in `lm_twoqtl` have been made to improve clarity. 'MC' in parameter statements now refers only to MCMC, with 'realizations' or 'simulation' being used in other cases.

Finally, for version 3.4 (December 2017). `simped_fgl` is replaced by a completely new `ibddrop` program, which simulates descent of founder genome labels (FGL) at dense locations, by simulating breakpoints of chromosome segments. Analogous changes for the use of dense markers were made to the `markerdrop` program. New features of `ibddrop` allow selection to be imposed at a specified locus, and a new beta-test program `ibd_trios` in the `IBD_haplo` directory allows the analysis of parent-offspring trios for detection of possible causal regions for inbreeding depression. Additionally, in all the `Lodscore` programs, specification of liability penetrances and age-dependent penetrances is much improved, and realization quantiles were added to the output of the `gl_lods` program. A new 'select all markers except ...' statement was implemented. The graphics option in `lm_auto` is no longer supported.

Combined with hands-on examples, this tutorial gives a brief introduction to the usage of the main MORGAN programs. For further information, please refer to the MORGAN documentation and to the references cited. Note also that the test Gold standards also provide many examples of parameter files. However, the Gold standards use short runs of MCMC and default seeds, and so should not be taken as useful for real analyses in this regard.

See [Concept Index], page 151, for: MORGAN, overview of MORGAN.

## 1.2 Get the Tutorial

This tutorial is available on-line at
`http://www.stat.washington.edu/thompson/Genepi/MORGAN/Morgan.shtml#tut`

Several formats of this tutorial may also be available to download for off-line reading or printing. (These may not all be made available for all releases; the html, PDF, and plain text versions are recommended.)

- Single HTML file (`http://faculty.washington.edu/eathomp/Anonftp/PANGAEA/MORGAN/morgan3-tut/morgan-tut.html`).
- Gzipped multiple HTML files (`http://faculty.washington.edu/eathomp/Anonftp/PANGAEA/MORGAN/morgan3-tut/morgan-tut.zip`) (one file per section).
- Hyperlinked PDF file (`http://faculty.washington.edu/eathomp/Anonftp/PANGAEA/MORGAN/morgan3-tut/morgan-tut.pdf`).
- Gzipped Postscript file (`http://faculty.washington.edu/eathomp/Anonftp/PANGAEA/MORGAN/morgan3-tut/morgan-tut.ps.gz`).
- Gzipped info file (`http://faculty.washington.edu/eathomp/Anonftp/PANGAEA/MORGAN/morgan3-tut/morgan-tut.info.gz`).
- Plain text file (`http://faculty.washington.edu/eathomp/Anonftp/PANGAEA/MORGAN/morgan3-tut/morgan-tut.txt`).

See [Concept Index], page 151, for: how to get the tutorial.

## 1.3 Get and set up the examples

This tutorial assumes that the MORGAN software has already been installed. If this is not the case, please contact your local system administrator or download the software yourself and follow the instructions therein.

Follow the following steps to download and set up the examples:

1. Download the examples (gzipped tar files) for MORGAN 3.4. morgan34-examples.tar.gz (`http: / / faculty . washington . edu / eathomp / Anonftp / PANGAEA / MORGAN / morgan34-examples.tar.gz`)

   (Note these examples are based on those for MORGAN 3.3., but have been updated for MORGAN 3.4 where at most minimal update is required. Other Examples have been removed for the Examples directory and from the tutorial text.).
2. Unpack the examples by typing the following command in a shell window,

   ```
   tar zxvf morgan3-examples.tar.gz
   ```

   Or if the above command fails (you don't have GNU tar), use

   ```
   gunzip -c morgan33-examples.tar.gz | tar xvf -
   ```

   This will produce a `MORGAN_Examples` directory under your current directory.

   (Note: Throughout the text, file and directory names are enclosed in single quotes; these single quotes are not part of the file or directory name.)
3. Use `Makefile` to establish links under the `MORGAN_Examples` directory to the MORGAN programs. A link under the `MORGAN_Examples` directory serves as a shortcut to a MORGAN program installed elsewhere.

   Before making links, you first need to edit the `Makefile` (using your favorite text editor, for instance `vim` or `nano`) in the `MORGAN_Examples` directory to make sure the paths to your MORGAN programs and those to the `MORGAN_Examples` directory are correct. Most often, it is necessary to change the '`MORGANDIR`' and '`EXAMPLEDIR`' statements to reflect the locations of the MORGAN files on your system and the examples, respectively. Here is the relevant part of the `Makefile`,

   ```
   # Change the following macros to where MORGAN and the examples
   # are installed on your system.  This is the only change you
   # need to make in this file.

   MORGANDIR = ~/morgan/MORGAN_V34_Release
   EXAMPLEDIR = ‘pwd‘
   BINDIR = ~/bin

   # Note: the paths may happen to be same for MORGANDIR and
   # EXAMPLEDIR. In general they are different:
   #   MORGANDIR is where MORGAN is installed on your system
   #   EXAMPLEDIR is the MORGAN_Examples directory you have made
   #   (we have used the BASH command ‘pwd‘ to automate this)
   #   BINDIR is your bin directory
   # BINDIR is needed only if you prefer to link to executables from
   # your bin directory, rather than running from executables in
   # a current directory.
   ```

   For more information on how to use Makefile to build links, etc., you may type:

   ```
   make help
   ```

   To make symbolic links to those programs in the current directory, type

   ```
   make links
   ```

*Notes for Microsoft Windows users*:

MORGAN may be (in principle) installed under Windows: executables should then be placed in the directory in which programs are to be run. See the documentation for more information. We cannot currently answer any questions regarding Windows installation. Instead, we recommend the use of a linux-system emulator such as Cygwin (`http://www.cygwin.com`).

See [Concept Index], page 151, for: how to get the examples.

## 1.4 Overview of the pedigrees used in the examples

Except for some small pedagogical pedigrees for `pedcheck` under `Pedcheck`, two main pedigree files are used to illustrate the usage of MORGAN programs.

File `jv_rep.ped`, located under `IBD`, is composed of two replicates of the JV pedigree. The 15-individual 5-generation JV pedigree derives from a real study of a rare recessive trait by Goddard et. al. [GYO96].

The other pedigree in `ped73.ped`, located under `MORGAN_Examples`, consists of three components and 73 individuals: component one has 47 individuals over 6 generations, component two 11 individuals over 3 generations, and component three 15 individuals over 3 generations. In general, individuals from later generations are observed. The three components are displayed in `ped47.pdf`, `ped11.pdf` and `ped15.pdf`, which are located in the subdirectory `PedInfo`.

The pedigree file `ped73.ped` is used in many examples throughout the tutorial: including Section 5.2 [Sample genedrop parameter file], page 30, Section 9.4 [Sample gl_auto parameter file], page 69, and the lod score programs Section 11.2 [Sample parameter files for lm_linkage and lm_bayes], page 91.

See [Concept Index], page 151, for: examples using pedigree file `ped73.ped`.

## 1.5 Structure of the MORGAN package

It is not necessary to read this section in order to use MORGAN, to run the examples, or to modify them for your own use. However, for those who wish to modify MORGAN code, or to understand MORGAN more fully, it will be useful to have information on the directory structure, the README documentation, and the GOLD-standard documentation, Makefiles, and examples. These are therefore described in this section, updated for the released version of MORGAN 3.4.

1. README documentation files

   Within the main MORGAN directory, there are program directories, and within these there the Gold-standard directories. At each level there are README files which provide additional documentation. In many cases, this information is duplicated in the tutorial, but whereas the Tutorial is focused to the user, README documentation is focused to the modifier and developer.

   1. README files in the main MORGAN directory

      These include `README_readme`, `README_MORGAN`, `README_install`, and `README_relnotes`.

      - `README_readme` describes the various README files throughout MORGAN.

    - `README_MORGAN` lists the MORGAN programs and describes briefly the analysis done by each program. It also lists the MORGAN 3.4 directories and libraries.
    - `README_relnotes` contains a summary of the changes and additions in recent releases of MORGAN.
    - `README_install` contains instructions for installing MORGAN executables.

   In some MORGAN releases there may be additional main-directory README files.
2. README files in main program directories

   The main program directories of MORGAN 3.4 are PedComp, Genedrop, Autozyg, Lodscore, IBD_Haplo, and PolyEM. Each main program directory contains its own `README_userdoc`. This describes the inputs to be prepared for the programs, and the various program options. Most of this information is now included in the tutorial, but the README files may contain more detail in some cases.

   Each of the main program directories Genedrop, Autozyg and Lodscore contains a file `README_convert2_3.0` which specifies the changes one must make in order to convert parameter files from MORGAN 2.9 to 3.0.

   The Library Subroutine directories do not contain README files.
3. README files in Gold and Test subdirectories.

   Each main program directory contains a subdirectory Gold. These directories include examples that may be run to check correct installation of MORGAN, and to provide a wider array of example parameter files than are currently in the example files used in the tutorial. Each Gold subdirectory contains a `README_gold` file detailing the examples in that directory.

2. The subroutine library directories

   The subroutine library directories contain the code for the library routines. During installation of MORGAN, each creates a library file from which the required subroutines are loaded into the executable of each main program.

   The header files for all libraries and programs are contained in the Headers subdirectory of MORGAN. Typically there is one or more header files associated with each library, and named accordingly. For example, the file `nghds.h` in `Headers` corresponds to the Nghds subroutine library. More complex libraries such as Pars have a large number of corresponding header files.

   The libraries can be divided broadly into four groups:
   1. Lowest-level libraries required by all programs
      - Stuff: Routines for printing, allocating, freeing
      - Pars: Routines for processing MORGAN parameter statements
   2. Low-levels libraries performing various groups of functions
      - CMF: A set of routines mainly for matrix manipulation, originally translated from the FORTRAN CM library
      - Peel: Routines for pedigree peeling computations
      - Rans: Routines for random number generation
   3. Main libraries supporting genetic analysis programs
      - Pedchk: Routines for checking validity of input pedigrees. Routines for the pedchk program in PedComp, but also called by all programs.

   - Nghds: Routines for constructing the pedigree neighborhood structures from input pedigree files. Used by all programs with input pedigree data files.
   - Quant: Routines for handling quantitative trait data. Used by PolyEM programs and others that use quantitative trait data. Relies on the CMF matrix manipulation library.
   - Markers: Contains routines for sorting and analyzing marker and trait data. Also all the routines that allocate and set the underlying inheritance vector arrays used by MCMC-based programs.
   - Sample: Routines for MCMC sampling and related computations on pedigrees.
4. Extra specialist libraries
   - TwoQTL: Routines for the Lodscore program `lm_twoqtl`
   - IBDgraph: Library to find equivalent IBD graphs; used by the Lodscore program `gl_lods`, and also by the `ibd_class` utility.
   - IBD_Create: Library containing the six subprograms of the `ibd_create` suite of programs, for use in simulating *ibd* graphs and structures, and haplotype data that can then be used in test analyses of other programs.

   In addition to the subroutine libraries, the subdirectory `Utils` of Autozyg contains code for subroutines that are directly incorporated into the `lm_ibdtests`, `lm_map` and `civil` programs. Also, the subdirectory `NewRtnes` of Lodscore includes code directly incorporated into the `lm_bayes` program. These routines were written by the authors of those programs, and have not been incorporated into the MORGAN subroutine libraries.

3. The main program directories

   The main program directories of MORGAN 3.4 are PedComp, Genedrop, Autozyg, Lodscore, IBD_Haplo and PolyEM. When MORGAN is installed these directories contain the following executables:
   - PedComp: pedcheck, kin, translink, ibd_class
   - Genedrop: genedrop, ibddrop, markerdrop
   - Autozyg: lm_auto, gl_auto, lm_pval, lm_ibdtests, civil, lm_map
   - Lodscore: lm_linkage, lm_bayes, lm_twoqtl, gl_lods, base_trait_lods
   - IBD_Haplo: ibd_create, ibd_haplo, ibd_trios
   - PolyEM: univar, unibig, bivar, multivar

   More details about all of these executable programs can be found either in this tutorial or in the README_userdoc files of the relevant main program directory.

4. GOLD-STANDARD directories, Makefiles and examples

   The Gold subdirectories of the main program directories PedComp, Genedrop, Autozyg, Lodscore, IBD_Haplo and PolyEM contain example runs of all the main programs in order to test various aspects of code and installation. Examples for a particular main program are in the Gold subdirectory of that main program directory.

   The Gold subdirectories typically contain numerous test parameter files, pedigree files, and marker data files. The tests are run via Makefiles, and the command `make help.gold` will provide details. Additionally, the `README_gold` file in each directory will give details of the examples.

Examples may run using the `make` command. Typically the complete set of examples in any Gold directory is run using the command `make all.gold`. More detailed information is given by using `make help.gold` or by viewing the `Makefile`. Since the Gold tests and examples are intended primarily for developers, it is expected that viewing and modifying the Makefile examples will pose no difficulties.

See [Concept Index], page 151, for: MORGAN package structure, README documentation files, MORGAN program libraries, MORGAN subroutine libraries, MORGAN Gold standards.

# 2 Common Features and File Formats

All MORGAN programs use the same command line syntax, share many statements, and use the same pedigree data format. Most of the MORGAN programs need at least two input files in order to run: one parameter file and one pedigree data file. The parameter file contains computing requests, model parameters and input/output file options. It may also contain genotype data or other information specific to a particular MORGAN program. The pedigree file contains, at minimum, information on family relationships among the individuals in the sample. If the general syntax and format descriptions of this section seem complex, readers may find it easier to proceed to the actual examples of the following chapter. In the context of those examples, the general format may become clearer.

It is worth pointing out that white space in any input file is defined to be any of these characters: '`,`' (comma), '`\t`' (horizontal tab), '`\v`' (vertical tab), '`\n`' (line feed, or newline), '`\f`' (form-feed), '`\r`' (carriage return).

See [Concept Index], page 151, for: whitespace.

## 2.1 Command syntax

The parameter file name must be passed to MORGAN on the command line when calling the program. Other file names can be passed to MORGAN on the command line or in the parameter file. The minimum syntax to call a MORGAN program is:

```
./progname parfile
```

In the statement above, *progname* is the name of one of several MORGAN main programs, such as `genedrop` or `lm_bayes`. The *parfile* is the name of the parameter file which must be present. For example, to run `genedrop` using a parameter file named `genedrop.par`, the command is:

```
./genedrop genedrop.par
```

Note that if the current directory is in your PATH, you may say

```
progname parfile
```

but the form *./progname* is more universal, and used throughout this tutorial.

Additional file names can be passed to MORGAN on the command line, but these file names must be accompanied by a file type to identify them. The syntax is:

```
./progname parfile [filetype filename]...
```

Square brackets indicate optional arguments. Possible *filetype* options include:

| | |
|---|---|
| `ped` | Input pedigree file |
| `xtra` | input extra file |
| `mark` | Input marker data file (Note that not all programs use marker data) |
| `oped` | Output pedigree file |
| `seed` | Input seeds for random number generator |
| `oseed` | Output random seeds |
| `oscor` | Output score file |

| | |
|---|---|
| `oxtr` | Output extra file |

If the name for a particular filetype is given both in the command line and in a parameter statement, the name in the command line takes precedence.

The programs put informational messages to `stdout` and error messages to `stderr` which default to the screen. It is possible to redirect either or both to a named file.

See [Concept Index], page 151, for: MORGAN files, command syntax, command line options, filetype codes.

## 2.2 Parameter file

A MORGAN parameter file contains a series of *statements*. Many statements are common to all MORGAN programs, particularly those that define the format of the pedigree file and identify other files to be used for program input or output. Many statements are optional, with some default behavior. If statements irrelevant to the MORGAN program called by the user are included in the parameter file, those statements are ignored and a warning message is issued.

Each statement must begin on a new line and begins with one of the MORGAN statement keywords. A statement consists of any number of lines. Case is not significant for the keywords. Only the first four letters of the keywords are significant; the remainder of the word is ignored. The order of the statements does not matter. If the same statement is repeated, the last one overrides previous ones and a warning is given in the output file. A `#` starts a comment so that the rest of the line is ignored. Either single or double quotation marks (`'` or `"`) can be used to delimit strings such as file names. Look at the warnings issued by MORGAN to make sure the parameters are as you intended.

Note that the parameter statement form is extremely flexible. There is a limit on the line input buffer (set to 30,000 characters), but this does not impose any restriction as a statement may extend over multiple lines. The parameter files described in this tutorial and MORGAN Examples and Gold standards are generally aligned and words stated in full for clarity, but this is not necessary. As an example, a parameter file for the `lm_linkage` program is given both in clear form and edited to show this flexibility in Section 11.2 [Sample parameter files for lm_linkage and lm_bayes], page 91.

The most common statements are for identifying input and output files (counterparts of the command line options) and for describing the input pedigree file format.

Below is a simple parameter file, `check.par`, from the examples included with the MORGAN software under the subdirectory `MORGAN_Examples/Pedcheck`.

```
set printlevel 5
input pedigree file ‘check.ped’
input pedigree size 30
input pedigree record gender absent
input pedigree record observed present
assign gender
output pedigree chronological
output overwrite pedigree file ‘check.oped’
output overwrite individuals file ’indiv_oped’
```

A brief description of the most commonly used parameter file statements follows in the next section. For a complete and more detailed description of MORGAN statements, please see the sections of this tutorial relevant to specific MORGAN programs and the documentation that comes with MORGAN in the files `README_userdoc` in the various program subdirectories.

See [Concept Index], page 151, for: line length limit, parameter file, parameter statements, parameter statement flexibility.

## 2.3 File identification statements

Within the parameter file, file names are delimited with single or double quotation marks ( ‘ or "). File names submitted on the command line are not delimited with quotation marks. In a parameter file, either of the two statements below would identify `pedchk.ped` as the pedigree file to be read.

```
input pedigree file "pedchk.ped"
input pedigree file ‘pedchk.ped’
```

The most commonly used file identification statements are:

`input pedigree file` *`filename`*
: The input pedigree file is required for most programs and may be specified either in the parameter file or through command line options.

`input extra file` *`filename`*
: The input extra file is used by some programs to input additional information, typically information needed by the program but for which parameter statements have not yet been implemented.

`input individuals file` *`filename`*
: Several newer MORGAN programs do not use a pedigree file but may still require a list of the individuals to be used in the analysis. Such a list is provided by the individuals file. See Section 2.8 [Individuals file], page 15.

`input marker data file` *`filename`*
: Marker data, such as marker allele frequencies, map distances between markers and individuals' genotypes, can be included in the parameter file itself or in a separate file, called the marker data file. This statement is used when the marker data are not included within the parameter file. The marker file contains the ‘`set marker data`’ statements. Marker data are used by Autozyg programs. See Section 9.8.7 [Autozyg computational parameters], page 77.

`input seed file` *`filename`*
: This file contains statements to set random seeds for the Monte Carlo based programs. The seed file may contain multiple lines (as in the case when the input seed file is also used for the output seed file). If so, the seeds in the last line override previous ones (with warnings issued). If no seed file is named on the command line or in a parameter statement and there are no statements to set random seeds in the parameter file, default seeds (12345, 1073 (hexadecimal 0x3039, 0x431)) are used.

`output [overwrite] pedigree file` *`filename`*
: The output pedigree file is required by `genedrop`. Other programs also check for errors in the pedigree. If there are errors that the program is able to correct

or if there are requested changes to the pedigree file format, the new pedigree data is written to this file.

`output [overwrite] individuals file` *`filename`*
: An individuals file may be created for those downstream programs that require is by using the output individuals file option out of the `pedcheck` program. See Section 3.4 [Creating an individuals file], page 20.

`output [overwrite] seed file` *`filename`*
: The final random seeds are saved if an output seed file is named. This file could be the same as the input seed file. New entries are appended to the old file, unless the overwrite option is specified.

`output [overwrite] score file` *`filename`*
: The output score (or scores) file is used by several programs to output numerical results, typically in a format for input to another analysis program.

`output [overwrite] extra file` *`filename`*
: The output extra file is used by some programs to output additional results.

Note that with MORGAN 3.0 several overwrite options have been added for output files, including pedigree and output scores files. Previously output scores were appended to existing output, if the file already existed, leading to confusion. This remains so, unless the overwrite option is used. Users should be cautious is using (and in not using) the overwrite option, and should, if using the option, be careful to copy previous output to another filename should they wish to retain it.

See [Concept Index], page 151, for: file names, pedigree file, individuals file, extra file, overwrite file options, marker data file, seed file.

## 2.4 Limit overriding

Many global constants are set in the header file `limdefs.h` in the Headers directory. Limits on variables may be altered by editing this file, but caution is recommended! Other limits may be set in other header files, for example `parseprog_opts.h`, and may be program-specific.

Additionally, there are three parameter statements which allow overriding of the preset limit values for the pedigree and trait-data file:

`allow component size` *`N`*
: This statement overrides the program-defined maximum pedigree component size (presently 400 individuals for most programs).

`allow observed individuals` *`N`*
: This statement overrides the program-defined maximum number of observed individuals; this applies only to some programs.

`allow pedigree size` *`N`*
: This statement overrides the program-defined maximum pedigree size (presently 20,000 individuals for most programs).

Finally there are three '`limit`' statements, relating to specific programs which will be detailed in the relevant chapters (Chapters 12 and 13).

See [Concept Index], page 151, for: limit overriding, limit statements, allow statements,

## 2.5 Input extra variables

There are two statements that are provided as development tools. These allow the user to input any number of integers and any number of reals into the program. These variables are counted (counts in global variables '`NumbDummyInts`' and '`NumbDummyReals`') and placed in global vectors '`DummyInts`' and '`DummyReals`'. The values of these integers/reals can then be accessed in any program.

These dummy variables should not be confused with '`dummy statements`' in some parameter files. These are statements which not relevant to the program in question, but are required in the parameter file in some programs which have been developed from other programs which use the statements.

`set dummy integers` *`I1 I2`*`...`
: Any number of integer variables may be entered into the program using this statement

`set dummy reals` *`X1 X2`*`...`
: Any number of reals variables may be entered into the program using this statement

See [Concept Index], page 151, for: dummy statements, dummy variables

## 2.6 Output control

By default, MORGAN sends its main output to standard output, `stdout`, and most warning and error messages to standard error, `stderr`. By default, both these will go to the terminal screen. Some programs use additional output files, such as the `output score file` or `output extra file`, to produce additional output, typically in a format for input for subsequent analyses.

Standard output may be redirected to a file, using the '`>`' symbol. For example

```
./genedrop genedrop.par > output-filename
```

The way in which standard output and standard error may both be redirected to the output file depends on the shell in use, but typically the '`>&`' redirect, or something similar, should work. For example

```
./genedrop genedrop.par >& output-filename
```

It is strongly recommended that users study the output warnings (W) produced, to check the program is interpreting parameter statements as expected.

Additionally, the standard output from each program is controlled by the following statement:

`set printlevel` *`N`*
: The level of output produced by all MORGAN programs is controlled by a printlevel ranging from 0 to 5. The value 5 leads to full output. For larger runs, particularly with large numbers of genetic markers, the user may prefer to suppress some output. It is recommended that users initially run their test data with printlevel 5, to check their input is being interpreted as expected.

While the `set printlevel` statement may be used to suppress unwanted output, the `set debug` statements can be used to obtain additional output. These statements are available for all main programs and libraries, but their result depends on what has been coded by developers in checking the software. The statements are intended primarily for developers, not the general user.

`set debug main`
: The set debug main statement applied in each main program to print additional information to `stdout` as coded in that specific program. Some programs may contain no such additional debug output code.

`set debug` *`libname`*
: The set debug *libname* statement is available for each library, where libname is one of **`cmf, ibdgraph, markers, nghds, pars, pedchk, peel, quant, rans, sample, stuff`** or `twoqtl`, corresponding to the relevant library name. The statement will cause additional information to be printed to `stdout` as coded in the subroutine files of that specific library.

`set debug level`
: The set debug level statement can be used in any program to, in principle, set the required level of additional debugging statements. However, currently only the `lm_twoqtl` program and `TwoQTL` library include code making use of the debug level.

In some earlier releases of MORGAN run-time display was available for the `lm_auto` program, using the GLUT library system, and the MORGAN library, `GLDisp` identified as `gldisp`. The display output is controlled by a set of 6 `display` statements. The run-time display, while in theory operational if GLUT libraries are installed, are not currently maintained, and are omitted from the current tutorial.

See [Concept Index], page 151, for: output control, debug control, printlevel control, output –redirect, output warnings (W), display options, display –GLUT runtime.

## 2.7 Pedigree file

The pedigree file may contain two sections, formatting statements and pedigree data, separated by the file separator '`****`'. The first section is optional; if present, it contains statements that describe the contents and format of the pedigree file, as many MORGAN users find it convenient to describe the pedigree data within the pedigree file itself. The alternative is to put these formatting statements in the parameter file.

The pedigree data begin below the file separator. Data for each individual must be placed on a separate line. Each line begins with three names, followed by integers, then real numbers. The only required fields, the three '`names`', are identifiers for each individual and his or her parents. Names may include up to 15 alphanumeric characters.

Whitespace (comma, space, tabs, linefeed), single (`'`) and double (`"`) quotes, and the hash mark (`#`) cannot be included in names. Names longer than 15 characters are truncated to 15 characters. Pedigree founders should be given parents with names '`0`'.

Gender, if present, is the fourth item in each line. Gender is coded as an integer, such that '`1`', '`2`' and '`0`' represent male, female, and unsexed, respectively.

These three or four values may be followed by an "observed" indicator, with values of '`0`', indicating an unobserved individual, or '`1`', indicating an observed individual. The optional "observed" indicator is followed by other integers, if present, and real numbers, if present. Integers and real numbers can represent individuals' trait data, covariates, or other information (for example, year of birth).

The format of the file is flexible and is specified by the user with '`input pedigree record ...`' statements, described in the next section.

Unlike LINKAGE format pedigree files, marker genotype data are not included in a MORGAN pedigree file.

See [Concept Index], page 151, for: pedigree file, parameter statements in pedigree files, pedigree file separator, whitespace, individuals –names, observed individuals, unobserved individuals.

## 2.8 Individuals file

Several newer MORGAN programs do not use a pedigree file, but still require a list of the individuals to be used in the analysis, potentially together with information such as gender, other covariate information or trait values. One such program is `ibd_haplo` which uses marker data to infer IBD using a population-based model. See Chapter 12 [Population-based inference of IBD], page 117.

Another program using the individuals file is `gl_lods`. See Section 11.6 [Parameter files for the gl_lods program], page 101. In this program, *ibd* graphs inferred from marker data using the `gl_auto` program are used to provide lod scores, given a trait model and trait data on some of the individuals in the *ibd* graphs. The goal is first to enable analysis of multiple trait models and even traits without re-running marker-based MCMC. Second, data security is enhanced by separating the pedigree information from the trait data.

The individuals file has similar format to the pedigree file, except that father and mother specifications for each individual are replaced by a component indicator. Since *ibd* graphs are generally produced by component, the `gl_lods` program does require this specification of pedigree component.

To assist users, the individuals file may be produced using the `pedcheck` program, using, for example, the same pedigree specification as is used for `gl_auto`. This will ensure consistency of component specification between the individuals file and the *ibd* graph input to `gl_lods`. See Section 3.4 [Creating an individuals file], page 20.

See [Concept Index], page 151, for: individuals file

## 2.9 Pedigree file description statements

Any of the following statements can be placed either in the parameter file or in the top section of the pedigree file, above the file separator, '`****`'. Most parameters have default values, in which case the statement is usually not required.

`allow pedigree size` *N*

This statement overrides the program-defined maximum pedigree size (presently 20,000 individuals).

`input pedigree size` *N*

Here, *N* is the number of records to be read. It may be less than the actual number of individuals in the pedigree file.

`input pedigree record names 3 [integers` *I*`] [reals` *J*`]`
: This specifies the numbers of entries in each line of the pedigree file. There must be three names (up to 15 alphanumeric characters each) identifying an individual and his or her parents. Optional integers include gender and phenotypic or discrete trait data. Real numbers could be covariates or quantitative trait values.

`input pedigree record (father mother | mother father)`
: This statement specifies the order of parental names. '`father mother`' is the default.

`input pedigree record gender (present | absent)`
: Gender, which follows the required triplet of names, is optional. If this statement is not included, the default is '`gender present`'. Gender is coded as an integer, such that '`1`', '`2`' and '`0`' represent male, female, and unsexed, respectively.

`input pedigree record observed (absent | present)`
: The observed indicator designates which members of the pedigree are observed and which are unobserved, indicated by '1' and '0', respectively. When the observed indicator is present, it follows gender (or parents if gender is not present). If this statement is absent, all pedigree members are assumed to be observed.

`input pedigree record traits` *K1 K2...* `integers` *I1 I2...*
: This statement is needed when integer data for traits are included, and the trait values do not immediately and consecutively follow gender (if present). Use this statement to specify the correspondence between trait numbers and integers in the record.

`input pedigree record traits` *K1 K2...* `reals` *X1 X2...*
: This statement is needed when real (non-integer) data for traits are included. The statement provides a correspondence between the trait and the column of the pedigree input file that contains those trait values. A real value with integer part 999 indicates a missing value.

Below are the first several lines of the sample pedigree file, `ped73.ped` in `MORGAN_Examples`.

```
input pedigree size 73
input pedigree record names 3 integers 7 reals 1

*************************************************
101 0 0 1 0 0 0 -1 -1 0 999.5
102 0 0 2 0 0 0 -1 -1 0 999.5
201 101 102 1 0 0 0 0 1 0 999.5
202 101 102 2 0 0 0 1 1 0 999.5
2010 0 0 2 0 0 0 -1 -1 0 999.5
301 201 2010 1 0 0 0 1 1 0 999.5
302 201 2010 2 1 3 2 1 1 0 105.945
304 201 2010 2 0 0 0 1 0 0 999.5
```

Note that marker genotype data are not contained in the pedigree file. These data, if required for the MORGAN program invoked, are contained in the parameter file or in a marker data file specified in the parameter file using the '`input marker data file`' statement. The second parameter statement in the file, '`input pedigree record names 3 integers 7 reals 1`' describes the format of the data on each line (also called a *record*) in the file. The first three values in each row, the names, give an individual's identification number followed by those of his or her father, then mother.

Because there is no '`input pedigree record gender`' statement, gender is assumed to be present and to directly follow the three names. Absence of an '`input pedigree record observed`' statement means that the `genedrop` program assumes all individuals are observed. This statement is not relevant to most other MORGAN programs although it can be used also by `pedcheck`.

The 6 integers following gender and the real number in the final column represent individual data. Lack of an '`input pedigree record traits integers`' statement would imply that the first integer following gender corresponds to trait 1, the second to trait 2, etc. However, these parameter statements are typically provided in the parameter file, not in the pedigree file.

These (and other) parameter statement defaults apply only if there is no overriding statement in any of the parameter files used. Programs will generally provide a warning statement (coded "(W)") when default values are being used due to absence of a relevant parameter statement.

See [Concept Index], page 151, for: pedigree file descriptions, pedigree size, pedigree record format, individuals –gender, pedigree trait data order.

# 3 Checking Pedigree Validity

## 3.1 Introduction to `pedcheck`

`pedcheck` reads the pedigree file and checks for errors in the pedigree structure. Specifically, it checks for the following errors:

- duplicate names of individuals
- individuals (non–founders) with parents missing from the pedigree
- individuals with parents of the wrong gender
- impossible pedigrees, such as an individual who is her or his own ancestor
- invalid names, genders, integers or real numbers
- pedigree entries with missing data

Note that, unlike some other packages, name identifiers must be unique across the entire data set, not only within pedigree components. If using pedigree files from other packages we recommend that new name identifiers be created if necessary, for example by combining pedigree (family) and individual identifiers: for example '`pedname_indname`'. In MORGAN names are arbitrary strings (subject to no whitespace) up to 15 characters in length to accommodate this translation easily.

If no errors are found, `pedcheck` reports the number of components (connected pedigrees) found and lists for each component:

- number of individuals
- the number of founders
- the number of females
- the number of males
- the number of unsexed individuals
- the number of observed individuals (if the '`observed`' indicator is present)
- the name of the first member of the component, in chronological order

If there are changes to the file, `pedcheck` writes an output pedigree file. Requested changes may include reordering of the pedigree chronologically (by component, then by individual), the addition of gender, the addition of an observed indicator, and reversing the order of the parental names.

Other MORGAN programs do their own pedigree checking by calling the relevant `pedcheck` functions, but it is still useful to do preliminary processing of data files first.

See [Concept Index], page 151, for: `pedcheck` introduction, pedigree validity, component, pedigree component.

## 3.2 Sample `pedcheck` parameter file

Files for `pedcheck` may be found in the `Pedcheck` subdirectory of `MORGAN_Examples`. Below is the sample parameter file `check.par` for `pedcheck`:

```
set printlevel 5
```

```
input pedigree file 'check.ped'
input pedigree size 30
input pedigree record gender absent
input pedigree record observed present
assign gender
output pedigree chronological
output overwrite pedigree file 'check.oped'
output overwrite individuals file 'indiv_oped'
```

The '`assign gender`' statement requests that `pedcheck` determine gender, when possible, and output that information to the output pedigree file. The gender determination is made based on the default order for the listing of parents, which is father followed by mother. Individuals who are not parents will be assigned missing gender, '`0`'.

'`output pedigree chronological`' causes the pedigree to be sorted into chronological order in the output pedigree file, first by component, then by individual name. MORGAN refers to each connected pedigree (i.e., distinct family) in a file as a *component*. The first individual in the input listing who is not genealogically connected to individual 1 defines component 2, and the first who is not connected to either of these defines component 3, etc. Although `pedcheck` groups individuals by their MORGAN–assigned component numbers in the output pedigree file, it does not list the component numbers. That is, the first three columns of the output file are just as they were in the input file: individual name, father's name, mother's name.

'`output overwrite pedigree file 'check.oped'`' specifies the output pedigree file. The overwrite option permits a previously existing `check.oped` to be overwritten. You should be cautious is using this option, in order not to overwrite files you wish to keep. However, if this option is not used, you will get an error message and the program will quit if `check.oped` already exists. If this occurs, you may delete the file and try again or use another output file name.

Finally, the '`output overwrite individuals file 'indiv_oped'`' in this version of `check.par` provides for the additional creation of an individuals file. See see Section 3.4 [Creating an individuals file], page 20.

See [Concept Index], page 151, for: `pedcheck` sample parameter file, component, individuals file, overwrite file options.

## 3.3 Running `pedcheck` examples

Examples for the program `pedcheck` are under the subdirectory `Pedcheck/`. The commands using example files are listed below. Have a look inside the pedigree and parameter files, then verify that the output files are as you would expect them to be. If error messages are generated, verify that they make sense and see if you can make the necessary corrections so that `pedcheck` will run.

*`./pedcheck check.par`*

runs on input pedigree file `check.ped`. The pedigree contains no errors, but has no gender specified and is not in chronological order. Look at the parameter file: you will see that it specifies the absence of gender, and requests that gender be assigned and that the output pedigree be chronologically ordered. Then, indeed, the output pedigree file `check.oped` has gender assigned and has the members reordered. Notice that individuals who are not parents (531 and 541) have missing gender, '`0`', in the fourth column of `check.oped`. The overwrite option permits a previously existing `check.oped` to be overwritten.

*`./pedcheck imp.par`*
: runs on input pedigree file `imp.ped`. The pedigree contains an individual who is his own ancestor.

*`./pedcheck empty.par ped sex.ped`*
: runs with an empty parameter file. The input pedigree file `sex.ped` is specified on the command line, and includes the `set printlevel 5` request. What does the output say is wrong with this pedigree?

*`./pedcheck empty.par ped dup.ped`*
: runs with an empty parameter file, with input pedigree file `dup.ped` specified on the command line. (in this case there is no `printlevel` statement.) What does the output say is wrong with this pedigree?

See [Concept Index], page 151, for: `pedcheck` examples.

## 3.4 Creating an `individuals file`

An individuals file may be created for those downstream programs that require it by using the output individuals file parameter statement. Inclusion of the statement

```
output overwrite individuals file "indiv.oped"
```

in any `pedcheck` parameter file will cause pedcheck to produce a list of individuals, their component numbers, and any integer and real covariate or trait information that was included in the input pedigree file.

If an individuals file with the name already exists, the program will append the output to the end of the file unless the overwrite option is specified. Since appending the output to a previous individuals file is probably not what the user wants it is highly recommended that the overwrite option is used.

The ordering of individuals is that of the output pedigree file (or input file, if this is unchanged): see Section 2.3 [File identification statements], page 11, and Section 3.2 [Sample pedcheck parameter file], page 18.

As an example you may add this line into the file `check.par` in the `Pedcheck` subdirectory of `MORGAN_Examples`. Compare the output individuals file `indiv.oped` with the output pedigree file `check.oped`.

## 3.5 `pedcheck` statements

`pedcheck` statements apply to other MORGAN programs since the programs call the `pedcheck` functions first to check the pedigree file before doing computations on the pedigree data.

- For specifying the pedigree file and the output pedigree file, see Section 2.3 [File identification statements], page 11.
- For describing the format of the pedigree file, see Section 2.9 [Pedigree file description statements], page 15.

`(assign | ignore) gender`
: Optional. '`assign gender`' causes gender to be determined by parentage, whether or not gender is included in the pedigree file. '`ignore gender`', causes the program to not check or assign gender. The default action is to assign gender when it is absent and to check gender if it is present.

`output pedigree chronological`
: Optional. If this statement is present and if the input file is not in chronological order, the pedigree is sorted and written out in chronological order. The pedigree is sorted by components, and within each component, each non-founding member is preceded by her or his parents. If this statement is not given, the input order is preserved in the output file, if written. See the previous section of this chapter for further discussion of pedigree components.

`output pedigree record (father mother | mother father)`
: Optional. This statement causes the parents to be named in the specified order. The default arrangement for each triplet of names is the input order.

`output [overwrite] individuals file` *`filename`*
: An individuals file may be created for those downstream programs that require is by using this `pedcheck` parameter statement. See Section 3.4 [Creating an individuals file], page 20.

See [Concept Index], page 151, for: `pedcheck` statements, `pedcheck` statements, pedigree options, individuals –gender, pedigree order, component, individuals file.

# 4 Computing Kinship and Other Pedigree Computations

## 4.1 Introduction to `kin`

`kin` computes kinship coefficients for pairs of pedigree members. It also computes single-locus and two-locus inbreeding coefficients for members of the pedigree. Briefly, the kinship coefficient between individuals *i* and *j* is the probability that a randomly-drawn allele from *i* is identical by descent (*ibd*) to randomly-drawn allele from individual *j* at the same locus. A single-locus inbreeding coefficient is the probability that an individual carries two copies of a gene that are *ibd*, at a given autosomal locus. In other words, an individual's single-locus inbreeding coefficient is equal to the kinship coefficient of his parents, as an individual's gametes can be thought of as random draws from his parents' chromosomes. A two-locus inbreeding coefficient is the probability that an individual carries two *ibd* copies of a gene at each of two linked loci. `kin` presents two-locus inbreeding coefficients as a function of the recombination fraction between the two loci.

Note: The `kin` program does check for duplicate requests within a pedigree component of any inbreeding or kinship coefficients; each will be computed once only, even if the individuals are specified in reverse order. However, it does check (and quits with an error) if a request is made for kinship of an individual with him/her self. This bug will be fixed in a future MORGAN release.

See [Concept Index], page 151, for: `kin` introduction, kinship coefficient, inbreeding coefficient, *ibd*,

## 4.2 Sample `kin` parameter file

Files for `kin` may be found in the `IBD` subdirectory of `MORGAN_Examples`. Below is a sample `kin` parameter file, `jv_rep_kin.par`.

```
set printlevel 5
input pedigree file 'jv_rep.ped'
compute component 1 kinship coeff           531 431   431 432
compute component 1 inbreeding coeff        332   531
compute component 2 kinship coeff           341  442
compute component 2 inbreeding coeff        441   541
compute component 1 two-locus inbreed coeff  531
compute component 2 two-locus inbreed coeff  441
set recomb freqs .01 .05 .04 .10 .18 .30 .50 .0
```

The statements on lines 3 – 8 request computation of kinship coefficients for the pairs '`531 431`' and '`431 432`', and then inbreeding coefficients for individuals '`332`' and '`531`', from component 1. It then requests kinship coefficients for the pair '`341 442`' and inbreeding coefficients for individuals '`441`' and '`541`' from component 2. Finally, it requests the two-locus inbreeding coefficient for '`531`' from component 1 and '`441`' from component 2. The two-locus inbreeding coefficient will be computed for two loci at distances specified in the '`set recomb freqs`' statement. (Note these need not be ordered, but the program will order them in the output.) If there is more than one component (connected pedigree) in the file, the component number must be specified. MORGAN assigns component numbers to the connected pedigrees within the pedigree file. If your data set contains more than one component, you may first run `pedcheck` to determine which individuals are assigned which component numbers.

See [Concept Index], page 151, for: `kin` sample parameter file.

## 4.3 Running `kin` example and sample output

Under the subdirectory `IBD/`, run the example above using the command below. To send the output to a file instead of the screen, include '`> filename`' (without quotes) after the parameter file name on the command line:

```
./kin jv_rep_kin.par    or
./kin jv_rep_kin.par > out-file-name
```

When you run this example you should get two warnings, '(W)', as there are two duplicate requests in the parameter file: `kin` recognizes these duplicates even though the order of individuals is reversed.

Below is the relevant part of the `kin` output.

```
Component 1:

   Kinship coefficients:

   531  431   .32031
   431  432   .10938


   Inbreeding coefficients:

   332   .00000
   531   .10938


   2-locus inbreeding coefficients:
   (g4link is probability of IBD at both of 2 linked loci)

   proband  recomb   g4link
              freq     prob

       531    .000   .10938
              .010   .10234
              .040   .08386
              .050   .07849
              .100   .05660
              .180   .03455
              .300   .01910
              .500   .01196


Component 2:
```

```
Kinship coefficients:

341  442   .15625


Inbreeding coefficients:

441   .06250
541   .10938


2-locus inbreeding coefficients:
(g4link is probability of IBD at both of 2 linked loci)

proband  recomb   g4link
           freq     prob

    441    .000   .06250
           .010   .05885
           .040   .04905
           .050   .04614
           .100   .03388
           .180   .02060
           .300   .01008
           .500   .00391
```

Note that when the recombination frequency is 0.0, the two-locus inbreeding coefficient is the same as the one-locus inbreeding coefficient, as there is no recombination between the loci, thus they act as a single locus. When the recombination frequency is 0.5, the two loci are independent and the two-locus inbreeding coefficient is the square of the one-locus inbreeding coefficient.

See [Concept Index], page 151, for: running `kin` example, `kin` sample output.

## 4.4 `kin` statements

At least one of the following '`compute ...`' statements are required to run program `kin`. If there is more than one component (connected pedigree) in the file, the component number must be specified. MORGAN assigns component numbers to the connected pedigrees within the pedigree file. If your data set contains more than one component, you may first run `pedcheck` to determine which individuals are assigned which component numbers. `pedcheck` will sort by component number in the output pedigree file, although it will not list component numbers in the file. The screen output generated when running `pedcheck` will give component numbers and the number of individuals in each component. Check the error and warning messages when running `kin` to verify that component numbers were correctly specified. The program will quit if an individual's component number is incorrectly specified in the parameter file or if there is more than one component in the data set and no component is specified.

`compute [component M] kinship coefficient N1 N2...`
: This statement names one or more pairs of pedigree members for which the kinship coefficient is to be computed.

`compute [component M] inbreeding coefficient N1...`
: This statement names one or more pedigree members for whom the inbreeding coefficient is to be computed.

`compute [component M] two-locus inbreeding coefficient N1...`
: This statement requests the computation of two-locus inbreeding coefficients, i.e. the probability of *ibd* at both loci, for the named individual. For the recombination frequencies at which the coefficients are computed, see the following statement.

`set recombination frequencies X1 X2...`
: Two-locus inbreeding coefficients are computed for each of the list of recombination frequencies, in the range of 0.0 to 0.5. If frequencies are not given, the default values are: 0.0, 0.01, 0.04, 0.05, 0.10, 0.18, 0.30, and 0.50.

See [Concept Index], page 151, for: `kin` statements, component, kinship coefficient, inbreeding coefficient, recombination.

## 4.5 The `translink` utility

The `translink program` converts MORGAN-style input files, including pedigree, trait, and marker data, marker map etc., into a LINKAGE-style input file.

The current format of the output dat file is for linkmap program. Since LINKAGE format requires integer IDs, if individuals are output by index, they are renamed 1 through n. However, recoding may still be required for some LINKAGE programs, if there are multiple component pedigrees.

The MORGAN statements used to specify the marker and trait information are those used by the '`Autozyg`' programs among others: see Section 5.4.2 [genedrop mapping model parameters], page 33. Note that because the program runs as an '`Autozyg`' programs various dummy statements not relevant to translink are required in the parameter file.

Here is an example of a `translink` parameter file:

```
# Version of pedigree and markers data for testing translink program.
# It includes 2 components, selection of markers, various traits.

# Include everything in the output file.
set printlevel 5

input pedigree file 'linkage.ped'
input marker data file 'linkage.markers'
input extra file 'translink.xtra'

select markers 3 4 6 7
select trait 1
set trait 1 tloc 6
set tloc 6 allele freqs 0.5 0.5

# ONE of the following three statements is required,
#  depending on the interval in which the trait locus is to be mapped
# map tloc 6 marker 1 recom frac 0.08   # First locus; final recom.
map tloc 6 marker 4 recom frac 0.01     # Within marker interval; "frac" ignored.
# map tloc 6 marker 7 recom frac 0.1    # Last locus; initial recom.

# Note; getting the left-side tloc position if markers are selected is a real
#   pain as specification is w.r.t original marker map.  Here 0.08 recombination
#   right of marker 1 is little bit left of marker 2, which is 10 cM from
#   marker 3: the first selected marker.
# Right hand side much easier -- just use last selected marker.

input pedigree record trait 1 integer 8
set trait 1 data discrete
set trait 1 for tloc 6 incomplete penetrances 0.05 0.6 0.95

# The following are DUMMY statements, required but irrelevant to translink
use single meiosis sampler
set MC iterations 1
set proband gametes 202 0 202 1
set sampler seeds  0x53f78285 0xdfbca001
```

The formats of these statements will be described in relevant sections later in the tutorial.

See [Concept Index], page 151, for: `translink` utility, LINKAGE package format, dummy statements.

## 4.6 The `ibd_class` utility

See [References], page 149, for details of the cited papers.

An *ibd* graph is a compact specification of the patterns of *identity by descent* among a set of individuals and across a chromosome [Tho11]. Chromosomal locations of changes in *bd* are indexed by integers; typically these are either marker or base-pair indices, The *ibd* graphs may be realized conditional on marker data (for example by using the program `gl_auto`), and used in subsequent analyses of trait data (for example by using the program `gl_lods`).

Normally the requirement of the trait analysis is to compute the probability of trait data given the *ibd* graph. However, on a given subset of individuals observed for the trait, many different realization of the graph, at many different marker loci, and even different graphs may give rise to identical trait probabilities. Recognizing when *ibd* graphs are equivalent in terms of their trait probabilities is key to efficient analysis: clearly probabilities should be computed once only for each set of equivalent graphs. The IBDgraph library, written by Hoyt Koepke [KT13], determines equivalence classes of graphs, across loci and across realizations.

The IBDgraph library can be accessed through the `gl_lods` program, but (from MORGAN 3.2.1) the utility `ibd_class` provides a direct standalone interface to the library.

There are five basic options in the ibd_class program. The following table described them using an input file of *ibd* graphs `subpedb.dat` in which there are are 1000 graphs and marker indices range from 1 to 201. The output is directed to `ibd_class.out` (appended in the case of examples after the first).

(1) `../ibd_class subpedb.dat -m 77 > ibd_class.out`

The `-m` option gives the equivalence classes at the specified index: here at marker 77.

The first part of the output for this example is

```
Testing at specified marker location 77.

1        : 254
1        : 414
1        : 639
6        : 9, 11, 105, 117, 122, 852
495      : 1, 3, 4, 17, ....
```

showing 3 classes of size 1, one of size 6, and then two large classes of size 495 and 496. Classes are listed in increasing size order.

(2) `../ibd_class subpedb.dat -r 156 160 >> ibd_class.out`

The `-r` options gives the classes that equivalent over a specified index range; here the range of markers 156 to 160. If the range is beyond the index range in the file, then the final graph specified (here marker 201) is assumed.

For this example the 72 classes range in size from 1 (24 classes) to 41.

(3) `../ibd_class subpedb.dat -s 155 99 >> ibd_class.out`

The `-s` options gives the range around the specified index (here marker 99) for which the specified graph (here graph 155) is invariant. For example, the result returned here is '`IBD graph 155 is invariant on the interval [86,116).`'

(4) `../ibd_class subpedb.dat -a 544 >> ibd_class.out`

The `-a` option gives all the index ranges over which the specified graph (here graph 544) is invariant. In this example the following output is produced:

```
IBD graph  544 is invariant on: [0,1),
                                [1,27),
                                [27,31), [36,57),
                                [31,36),
                                [57,78),
                                [78,86),
                                [86,112), [113,117),
                                [112,113),
                                [117,121),
                                [121,137),
                                [137,138),
                                [138,155),
                                [155,156),
                                                          [156,160),
                                                          [160,170),
                                                          [170,173),
                                                          [173,177),
                                                          [177,184),
                                                          [184,197),
                                                          [197,inf)
```

Note that the sets of indices need not be contiguous. In this example, the graph at markers 31 to 35 differs from same graph on each side, and the graph at marker 112 also provides a break to a different graph.

`(5) ../ibd_class subpedb.dat >> ibd_class.out`
: Without any option specified, the program produces the classes of graph that are equivalent across all indices. In this example. there are no graphs equivalent across all 201 markers, so the program produces a list of 1000 classes each of size 1.

`(6) ../ibd_class subpedb.dat -e >> ibd_class.out`
: The `-e` option is the most complex, giving the equivalence classes both across marker ranges and across graphs. The output is not easy to parse by hand, but it is the option required for most efficient use of the software. The program `gl_lods` uses this option to ensure that lod score contributions are computed once only for each equivalence class of graphs in the entire set.

See [Concept Index], page 151, for: IBDgraph library, `ibd_class` utility, *ibd* graph

# 5 Simulating Marker and Trait Data in Pedigrees

See [Concept Index], page 151, for: simulating marker and trait data.

## 5.1 Introduction to `genedrop`.

`genedrop` simulates pedigree data for analysis by other programs. Given a genetic map, it simulates genotypes at marker loci (linked or unlinked) and the discrete genotypes and polygenic values contributing to quantitative traits. The trait loci may or may not be linked to marker maps. Thus, one or more of three kinds of loci are simulated on a chromosome: markers, traits linked to markers, and traits not linked to markers.

`genedrop` assigns marker and trait genotypes and polygenic trait values to the founders by using a random number generator. Meiosis indicators are then simulated for non-founders in chronological order, thus determining the founder genome labels inherited. Markers and traits, if present, are then simulated for each individual: First, marker genes are simulated in the order mapped on the chromosome, then linked traits are simulated in map order, and finally, unlinked traits are simulated.

Because founders of a pedigree are assumed to be unrelated, a unique identifier or *founder genome label* is assigned to each of the two haploid genomes of each founder. The user may choose to identify the ancestral source of each gene at each locus in non-founders by including the founder labels in the output pedigree.

The user may provide random number seeds for both the marker simulation and the trait simulation. This permits multiple simulations, for a pedigree, of identical marker genotypes, but with different quantitative trait values.

The population and segregation model parameters (trait genotype means, additive and residual variances) may be specified by the user and take default values if not specified. Allele frequencies have no default values and must be specified by the user. Several different trait models can be specified as in the following table:

| | Equal Genotypic Means | Zero Additive Variance |
|---|---|---|
| *non-genetic model* | YES | YES |
| *polygenic model* | YES | NO |
| *major gene model* | NO | YES |
| *mixed model* | NO | NO |

The trait locus must have two alleles and the trait residual variance must be greater than zero. A very small residual variance can be specified if one desires to simulate a qualitative trait.

Genetic data on all individuals may be included in the simulated pedigree, or some individuals may be specified as '`missing`'. If any individuals are to be missing genetic data, an '`observed`' indicator column must be included in the pedigree file. See Section 2.7 [Pedigree file], page 14, for details.

See [Concept Index], page 151, for: `genedrop` introduction, quantitative trait, polygenic model, major gene model, mixed model, non-genetic model, founder genome labels, seeds for data simulation, additive variance, unobserved individuals.

## 5.2 Sample genedrop parameter file

Files for `genedrop` may be found in the `Simulation` subdirectory of `MORGAN_Examples`. The example here refers to `ped73_gdrop.par`.

The seed file is used to store the random seeds used in the simulations. Occasionally one will want to use the same seed with multiple runs, but most often one will want to use new seeds so as to obtain different output with each run. The seed file contains one or more statements like '`set marker seeds 0xde5e8d39`'. For more about the way `genedrop` handles seeds See Section 5.4.4 [genedrop computational parameters], page 35.

The seed file can be specified in the command line or in the parameter file. The following statements are needed to specify the seed file in the parameter file:

```
input seed file '../marker.seed'
output marker seeds only
output overwrite seed file '../marker.seed'
```

The first line specifies `marker.seed` in the main examples directory as the input seed file for the marker simulation. The second statement, '`output marker seeds only`', overrides the default behavior of saving both the marker and the trait seeds and causes the program to save only the marker seeds before exiting. The `overwrite` option in line 3 enables the program to replace the current seed file content with the newly generated random numbers, which can be used for simulation in the future. When an overwrite is not requested, MORGAN appends the new output seeds to the existing file at the end of the run. Thus, at the next run, more than one '`set marker seeds`' statement exists in the seed file. The program uses only the last '`set marker seeds`' statement in the file.

In the example, we have chosen to access the seed file from the command line, which will overrule the parameter file statement and generate a warning. See the next section for command line implementation.

*Note:* The statement '`output pedigree chronological`' is included in the example `ped73_gdrop.par` file so that the output pedigree will be in the chronological order required for use with other MORGAN programs.

The next statements in the parameter file are the simulation requests:

```
simulate chrom 1 markers
simulate traits 1
set traits 1 tlocs 1
```

The above statement asks `genedrop` to simulate marker loci on chromosome 1. Additionally, one quantitative trait controlled by one tloc will be simulated. The number of markers, and the relative locations of tloc and marker loci will be determined from the '`map`' statements below. In MORGAN-3, traits are distinguished from trait loci, and thus the statement '`set traits 1 tlocs 1`' assigns trait 1 to trait locus 1. In general one or more traits may be assigned to any given trait locus. If no trait locus is to be simulated, the lines '`simulate traits 1`' and '`set traits 1 tlocs 1`' can be removed.

```
map chrom 1 marker dist  10 10 10 10 10 10 10 10 10
map chrom 1 tlocs 1 marker 5 dist 5
```

The above statement indicates a marker map on chromosome 1, with 10 equally spaced markers, each at a distance of 10 (Haldane) centiMorgans from the preceding one. Note that the number of markers is inferred from this statement. The trait locus is between markers 5 and 6 on chromosome 1, at a distance of 5 cM to marker 5.

A marker map or tloc position can also be specified by recombination fractions. For example,

```
map chrom 1 marker recomb fracs 0.1 0.5 0.2
```

gives a map of four ordered markers, M1,M2,M3 and M4, with recombination fraction 0.1 between M1 and M2, 0.5 between M2 and M3, and 0.2 between M3 and M4.

Marker allele frequencies are set by the following lines:

```
set chrom 1 markers 1  allele freqs 0.13 0.66 0.16 0.05
set chrom 1 markers 2  allele freqs 0.06 0.23 0.41 0.25 0.05
set chrom 1 markers 3  allele freqs 0.11 0.02 0.01 0.06 0.24 0.56
set chrom 1 markers 4  allele freqs 0.07 0.04 0.89
set chrom 1 markers 5  allele freqs 0.12 0.11 0.03 0.03 0.50 0.21
set chrom 1 markers 6  allele freqs 0.50 0.44 0.06
set chrom 1 markers 7  allele freqs 0.01 0.33 0.62 0.04
set chrom 1 markers 8  allele freqs 0.20 0.05 0.42 0.27 0.06
set chrom 1 markers 9  allele freqs 0.18 0.18 0.25 0.16 0.08 0.15
set chrom 1 markers 10 allele freqs 0.17 0.35 0.04 0.29 0.15
```

In the case where several markers have the same number of alleles and allele frequencies, one can group those markers together into one line:

```
set chrom 1 markers 11 12 13 15 allele freqs 0.2 0.8
```

However, we consider it good practice to specify the frequencies separately for each marker.

The following five lines describe the trait model. The trait locus can have only two alleles; here the frequencies are 0.5 and 0.5, for alleles 1 and 2, respectively. The mean values of the trait for each trait locus genotype are on the next line. Values correspond to the (1 1), (1 2) and (2 2) genotypes, respectively. The residual variance gives the within-genotype variance of phenotypic values about the mean. The additive variance (0 in this example, and by default if not specified) is the variance of an additive polygenic contribution to trait values.

```
set trait 1 allele freqs 0.5 0.5

set trait 1 for tlocs 1 geno means 90 100 110
set trait 1 residual variance 25.0
set trait 1 additive variance 0.0
```

The following three lines may be included in the parameter file (we have commented them out in the example so as to keep the output file small and easy to read).

```
output pedigree record founder genome labels
output pedigree record trait latent variables
output pedigree record unobserved variables
```

These lines request that the founder genome labels and latent variable values for the trait be included in the output file, and that the data be output for all (observed and unobserved) individuals. Founder gene labels indicate, for all non-founders, which founder alleles were passed to the individual. For the trait variables, the latent founder genome labels, the trait locus genotype, and the additive and residual contributions to the trait value are given. Latent trait variables will precede the trait value in the output file.

See [Concept Index], page 151, for: `genedrop` sample parameter file, seed file, additive variance, residual variance, founder genome labels.

## 5.3 Running genedrop examples and sample output

Two examples are available under the subdirectory `Simulation/`. The only difference is in whether command line options are to replace some parameter statements (see the `README` file in the `Simulation` directory).

The command to run the first example is:

```
./<program> <parfile> [ped <pedfile>] [seed <seedfile>] [oped <opedfile>]
./genedrop ped73_gdrop.par ped ../ped73.ped seed ../marker.seed oped gdrop.oped
```

For the parameter file `ped73_gdrop_2.par'` the three files (input and output pedigree, and seed) are given in the file, and so are not needed as command line options. This file may be run simply as

```
./genedrop ped73_gdrop_2
```

The output is the same as for `ped73_gdrop.par`.

When running the `genedrop` example, we here use an (unchanging) input marker seed file `../marker.seed` but output to the current directory file `marker.seed`. However, in practice a file such as `marker.seed` can be specified as both the input and output seed file. If a `overwrite` option if not included in the '`output seed file`' statement, successive runs will generate warnings (W), but this is not a concern. Recall from the previous section that, by default, MORGAN appends the new output seeds to the existing seed file at the end of each run. In the next run, the last (most recent) seed will be used. To avoid this warning (and an ever-growing seed file), either use the `overwrite` when outputting the seeds (see the previous section Section 5.2 [Sample genedrop parameter file], page 30), or manually edit the seed file removing earlier lines.

Since the function of `genedrop` is to simulate marker and trait data, it, unlike other MORGAN programs, always creates and outputs a pedigree file. The output file `gdrop.oped` is structured similarly to the input file `ped73.ped`, with one individual per record (line). However, the output file contains additional columns and does not include the parameter statements found at the top of the input file. The first four items are the individual's name, the names of the parents, and gender. If no addition output options are set, the next items are the genotypes of the markers (two items per marker) in the order they are found on the chromosomes, followed by the trait values in the order of the trait labels.

Notice the three statements at the end of the parameter file. In order to save space and make the output more readable, these statements have been commented out so that they are not executed by the program.

If the statement '`output pedigree record trait latent variables`' was included in the parameter file, the output file would contain four additional columns preceding the trait value. The first two of these columns would be the trait locus genotype, followed by the additive component of the trait value and the residual component of the trait value. In this example, everyone has a '`0.000`' in the additive component column because we set the additive variance to zero in the parameter file.

If the '`output pedigree record founder gene labels`' is set, the founder genome labels (FGL) for markers precede the marker genotypes and the trait FGL precede the trait values (or the trait latent variables, if these are requested).

Also, if the '`output pedigree record unobserved variables`' statement is included in `gdrop.par`, an observed indicator would follow gender in the output pedigree file. Also, marker and trait data would be output for all individuals, not only those indicated as '`observed`'.

See [Concept Index], page 151, for: running `genedrop` examples, `genedrop` sample output, seeds for data simulation.

## 5.4 genedrop statements

See [Concept Index], page 151, for: `genedrop` statements.

### 5.4.1 genedrop computing requests

`simulate [chromosome` *`I`*`] markers`

One statement is given for each chromosome on which markers or both markers and traits are to be simulated. Only unlinked traits are simulated if no such statement is provided. The '`chromosome`' keyword can be omitted if all markers and linked traits are on the same chromosome. Note that the number of markers is inferred from the number mapped on the chromosome in the parameter file.

`simulate traits` *`K1`*

The linked traits to be simulated are specified here. The linked traits are specified as positive integers.

`set traits` *`K1`*`... tlocs` *`L1`*`...`

This statement establishes the correspondence between traits and trait loci. Presently in `genedrop` each trait may have only one trait locus, but more than one trait may be assigned to the same locus. The trait loci are specified as positive integers.

`map tlocs` *`L1`*`... unlinked`

Optional. This statement specifies trait loci which are unlinked to specific traits, and hence have no map specification.

See [Concept Index], page 151, for: `genedrop` computing requests.

### 5.4.2 genedrop mapping model parameters

`map [chromosome` *`I`*`] [gender (F | M)] marker ( [Kosambi] distances | recombination fractions | [Kosambi] positions)` *`X1 X2`* `...`

This statement is required if simulation of more than one marker is requested. One statement is used per chromosome. This statement specifies the marker map or positions given in units of genetic distances (cM), or recombination fractions between markers. Marker map or positions can be sex-specific if gender is included in the statement. If '`distances`' is chosen, intermarker distances are provided such that the number of distances is one less than the number of markers. If '`positions`' is chosen, the number of positions is equal the number of markers, as these are absolute positions relative to a zero point to the left of all of the markers. The Haldane mapping function is used to convert between the genetic distances and recombination fractions unless Kosambi is specified.

`map [chromosome` *`I`*`] [gender (F | M)] tlocs` *`K1 K2`* `... markers` *`J1 J2`* `... ( [Kosambi] distances | recombination fractions)` *`X1 X2`* `...`

This statement is required if simulated trait loci are to be linked to markers; i.e., it is not required if no trait loci or only unlinked trait loci are to be simulated. The statement specifies the location of each trait locus with respect to one of the marker loci. Thus, the number of trait loci listed in the statement must be equal to the number of markers listed and to the number of distances (or recombination fractions) listed. The trait locus will follow the corresponding marker locus (to the right, so to speak) at the distance specified. To simulate a trait locus that precedes all marker loci, list marker '0' in the statement. For example, with '`map tlocs 3 2 marker 6 0 distances 5 4`', trait loci 3 and 2 will be placed 5 cM to the right of marker 6 and 4 cM to the left of marker 1, respectively.

See [Concept Index], page 151, for: `genedrop` mapping model parameters, gender–specific maps, Haldane map function, Kosambi map function.

### 5.4.3 `genedrop` population model parameters

`set [chromosome` *`I`*`] markers` *`K1`* `... allele frequencies` *`X1 X2`* `...`

This statement specifies markers allele frequencies. Allele frequencies for a marker should sum to between 0.9999 and 1.0001. Otherwise they are normalized. Multiple markers can be specified in a single statement if they reside on the same chromosome and have the same number of alleles with the same allele frequencies.

`set tlocs` *`K1`* `... allele frequencies` *`X1 X2`* `...`

This statement specifies the trait loci allele frequencies. Allele frequencies for a trait locus should sum to between 0.9999 and 1.0001. Otherwise they are normalized. Multiple trait loci can be specified in a single statement if they have the same allele frequencies. Trait loci must have two alleles.

`set normalized allele frequencies`

If the set of allele frequencies for each marker and trait is to be normalized, this statement is given. Normalization of the frequencies is recommended when simulating pedigree data, but not recommended when using the other programs.

`set traits` *`K1`* `for ... tlocs` *`L1`*`... genotype means` *`X1 X2 X3`*

Since two alleles are simulated for each trait locus, three means must be specified for the polygenic trait values: one each for the (1 1), the (1 2) or (2 1), and the (2 2) genotypes. The default values 0.0, 0.0, and 0.0.

`set traits` *`K1`* `... additive variance` *`X`*

Here we specify the genetic variance for one or more trait. One of there statements is given for each value assigned. The default variance is 0.0.

`set traits` *K1* `...` `residual variance` *X*
: This statement is like the preceding one. The environmental contribution to the trait is set using this statement.

See [Concept Index], page 151, for: `genedrop` population model parameters, allele frequencies, additive variance, residual variance.

### 5.4.4 `genedrop` computational parameters

`set marker seeds` *H1 H2*
: This statement initializes the seeds for the random number generator in the gene dropping algorithms. The seeds are to be positive and no greater than hexadecimal 0xFFFFFFFF, with the first seed (congruential seed) odd, and the second seed (Tausworthe seed) nonzero. In `genedrop`, markers are simulated before traits, so that, if no seeds are specified for marker simulation, default seeds (0x3039 0x431) are used.

`set trait seeds`
: *H1 H2* This statement initializes the seeds for trait simulation. If no seeds are given, the starting seeds for trait simulation are the seeds returned by the random number generator at completion of marker simulation. Note that if output of marker seed is requested, this will be the same value as is output to the marker seed file for a subsequent `genedrop` run.

See [Concept Index], page 151, for: `genedrop` computational parameters, seeds for data simulation, simulating marker and trait data.

### 5.4.5 `genedrop` output pedigree options

`output pedigree record founder gene labels`
: When this option is selected, each record contains a pair of founder genome labels for each locus. Each founder is assigned a pair of labels, which are in the same order as the names of the parents. Then, for each locus of each descendant, founder genome labels are determined by the simulated meiosis indicators.

: This statement is useful in cases where the founder origins or descent of trait locus alleles are required, for example in assessing the results of subsequent analyses of the simulated data.

`output pedigree record trait latent variables`
: This statement requests that the quantitative trait latent variables be included in the output. The genotype at each trait locus, as well as the additive and residual component of each quantitative trait, will appear in the output record.

`output pedigree record unobserved variables`
: If this option is set, genotypes, gene labels and trait values are output for both observed and unobserved individuals. An additional data field, following the gender indicator, specifies whether the individual is observed (‘`1`’) or unobserved(‘`0`’).

: When this option is not selected, unobserved individuals take on default values; the genotype at each locus represented as ‘`0 0`’, the founder genome label (if requested) at each locus represented as '`0 0`', and each quantitative trait value is recorded as '`999`'.

`input pedigree record observed (absent | present)`
: The observed indicator is used to designate which members are observed, with '0' indicating unobserved, '1' indicating observed. When the observed indicator is present in the pedigree file, it follows gender (or parents, if gender is not present). If this statement is not given, all pedigree members are assumed to be observed. See also the next statement '`assume all observed`'.

  If individuals are flagged in the pedigree file as unobserved, the default behavior is to indicate in the output pedigree file that the data for these individuals is missing.

`assume all observed`
: When this statement is used, all members of the pedigree are treated as "observed" in the simulation. If an observed indicator column is present in the input file, it is ignored by the simulation.

See [Concept Index], page 151, for: `genedrop` output pedigree options, founder genome labels, meiosis indicators, inheritance indicators, unobserved individuals.

### 5.4.6 `genedrop` output seed file options

`output (marker | trait) seeds only`
: If an output seed file is given, both ending marker and trait seeds are saved unless one or the other is requested in this statement.

See [Concept Index], page 151, for: `genedrop` output seed file, seeds for data simulation.

# 6 Simulating Marker Data Conditional on Trait Data in Pedigrees

## 6.1 Introduction to `markerdrop`

`markerdrop` simulates marker data at markers linked to a hypothetical trait locus. The user must specify whether marker data simulation is to be conditional on a trait model and trait data (the trait option) or as a (possibly partial) specification in the pedigree file of the inheritance at the trait location (the inheritance option). The choice of a trait model or an inheritance pattern will dictate which additional parameter statements must (or may) be included in the parameter file. For the trait option, the pedigree file contains trait data; for the inheritance option, the pedigree file contains meiosis (inheritance) indicators. See Section 8.1 [Specifying inheritance], page 56.

- If marker data simulation is to be conditional on a trait model, parameters must be provided for trait locus allele frequencies using '`set tlocs allele frequencies`', for genotypic penetrances using '`set trait for tlocs incomplete penetrances`', and for the map position of the trait locus using a '`map`' statement see Section 6.5 [markerdrop statements], page 43. There must be only one mapping statement for the trait locus; from this statement the trait locus number (name) is deduced. Phenotypic trait data are provided as affection status of each individual in the pedigree file. An inheritance pattern at the trait locus is simulated from the trait data; this becomes the trait model on which markers are simulated.
- If marker data simulation is to be conditional on an inheritance pattern at the trait locus, the partially specified segregation pattern at the trait locus is provided in the pedigree file using meiosis indicators. For more information on meiosis indicators, see Section 6.5.1 [markerdrop computing requests], page 43, and Chapter 8 [Using MCMC to Estimate Parameters of Interest in Pedigree Data], page 56. Location of meiosis indicators in the pedigree file can be specified using the '`input pedigree record`' statement. Again, specification of a map position for the trait locus using a '`map`' statement is required.

The program `markerdrop` first generates descent at the trait locus, and then, conditionally on this, descent at marker locations across the chromosome. At the requested locations, founder genome labels of observed individuals are determined, and then marker genotypes are imposed at each locus given the realized FGL and marker allele frequencies. In MORGAN V3.4, a new option to simulate 'dense markers' is provided. In this case, the simulation of descent across the chromosome, instead of being generated marker-ti-marker, is done by simulating recombination breakpoints, and an *ibd* graph; See Section 12.5 [Running ibd_create examples and sample output; simpop_fgl], page 123. Only the descent determination uses this algorithm; thereafter, at each requested marker, founder genome labels are determined and alleles and genotypes assigned as before.

See [Concept Index], page 151, for: `markerdrop` introduction, trait model, incomplete penetrances, meiosis indicators, dense markers, founder genome labels.

## 6.2 Sample `markerdrop` parameter file – conditional on trait

Files for `markerdrop` may be found in the `Simulation` subdirectory of `MORGAN_Examples`. The sample parameter file `ped73_mdrop_trait.par` requests simulation of marker data conditional on a trait model. The trait is assumed to be discrete when simulation is conditional on a trait model. Examples using the 'dense marker' option may be found in the '`Genedrop`' Gold standards.

Note that the `ped73_mdrop_trait.par` parameter file contains a '`set printlevel`' statement. MORGAN programs will produce varying levels of output given the print level. We recommend setting the print level to 5 for initial testing purposes. However, if the 'dense markers' option is selected, it is best to suppress printing of the marker map, by, for example, '`set printlevel 3`'.

Many of the statements for simulation of the markers conditional on trait data are similar to those used in `genedrop`: See Section 5.2 [Sample genedrop parameter file], page 30. However, rather than simulating trait loci or trait data, these are provided to the `markerdrop` program.

The relevant section of the file is:

```
simulate markers
select trait 2
set traits 2 tlocs 1
map marker positions 10 20 30 40 50 60 70 80 90 100
map tlocs 1  marker 5  dist 5.0

set trait 2 data discrete

set traits 2 for tlocs 1 incomplete penetrances 0.05 0.8 0.95
set tlocs 1 allele freqs 0.5 0.5

set markers 1 allele freqs 0.13 0.66 0.16 0.05
set markers 2   allele freqs 0.06 0.23 0.41 0.25 0.05
.
.
set markers 10 data

101 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
201 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
202 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2010 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
301 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
302 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
304 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
.
.
.
```

The first four lines are required for `markerdrop` and must be included in the parameter file. The '`map tlocs`' statement identifies the trait locus to be used in the simulation and gives its position relative to the markers on which we are simulating data. In this example, the trait locus follows marker 5 at a distance of 5 centiMorgans. The '`simulate markers`' and '`select trait`' statement indicates that the markers will be conditional on a trait model.

The '`map marker positions`' statement specifies the spacing of the markers to be simulated, from which the number of markers is also inferred.

Note that the parameter file for running a simulation conditional on a trait model requires two more lines than the parameter file for simulation conditional on an inheritance pattern (see next section). These two additional lines are required for discrete traits (the default for simulation conditional on a trait). The statement '`set traits 2 for tlocs 1 incomplete penetrances ...`' specifies the probability of exhibiting the trait for individuals with trait locus genotypes '`1 1`', '`1 2`' (or '`2 1`') and '`2 2`', respectively. The statement '`set tlocs ... allele freqs`' specifies trait locus allele frequencies.

The '`set markers...allele freqs`' statements be included; they specify allele frequencies at each markers.

The `markerdrop` program uses the '`set markers 10 data`' statement to specify which individuals and at which loci marker data are required. This is the same statement used by other programs in the analysis of marker data; see, for example Section 9.8.7 [Autozyg computational parameters], page 77. Marker data are specified for each marker locus as a pair of integer alleles, and '`0`' indicates a missing value. For `markerdrop` any non-zero value will indicate that the data are to be observed. Typically, one may enter regular marker data, in order to generate other marker data with the same missingness pattern. Alternatively, as here, a '`1`' may be used to indicate that the corresponding marker data are observed. Note that individual '`302`' is the first observed member in this data set, and is observed for all 10 markers.

The parameter file `ped73_mdrop_trait.par` uses the pedigree file `ped73.ped`, which is found in the `MORGAN_Examples` directory. The file format section and first few lines of the pedigree data section of this file are below.

```
input pedigree size 73
input pedigree record names 3 integers 7 reals 1

*****************************************************
101 0 0 1 0 0 0 -1 -1 0 999.5
102 0 0 2 0 0 0 -1 -1 0 999.5
201 101 102 1 0 0 0 0 1 0 999.5
202 101 102 2 0 0 0 1 1 0 999.5
2010 0 0 2 0 0 0 -1 -1 0 999.5
301 201 2010 1 0 0 0 1 1 0 999.5
302 201 2010 2 1 3 2 1 1 0 105.945
```

The first three columns are indices are 'names' which are character strings. They are unique identifiers of each individual and his/her parents. By default, the parent order is father followed by mother. The next four columns are sex (1=male, 2=female), observed status (0=unobserved, 1=observed) and possible trait or other data. Recall that in the parameter file we had

```
select trait 2
```

Now we see also in the parameter file

```
input pedigree record trait 2 integer 4
```

This statement specifies that '`trait 2`' is the fourth integer in the pedigree file, after the three names (that is, the 7 th item). Traits may be given any integer label: here '`2`' is an arbitrary choice. This column of the pedigree file contains the affection status for a discrete traits (0=missing, 1=unaffected, 2=affected).

If desired, this statement can be included in the pedigree file instead. Other columns is the pedigree file are explained in the next section.

Note that `markerdrop` can simulate data for markers linked to only one trait locus, as specified in the '`map`' statement in the parameter file.

See [Concept Index], page 151, for: `markerdrop` parameter file – conditional on trait, print-level control.

## 6.3 Sample `markerdrop` parameter file – conditional on inheritance pattern

The sample parameter file, `ped73_mdrop_inhe.par`, requests simulation of marker data conditional on an inheritance pattern. The relevant section of the file is:

```
simulate markers
select inheritance 1
set inheritance 1 tlocs 1
map marker positions 10 20 30 40 50 60 70 80 90 100
map tlocs 1 marker 4 recomb frac 0.01

set markers 1 allele freqs 0.13 0.66 0.16 0.05
set markers 2   allele freqs 0.06 0.23 0.41 0.25 0.05
.
.
.
set markers 10 data

101 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
201 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
202 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2010 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
301 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
302 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
304 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
.
.
.
```

The first five lines are required and must be included in the parameter file. The '`simulate markers`' and '`select inheritance 1`' statement indicates that we are simulating marker data conditional on inheritance, and identifies the inheritance pattern to simulate. The '`set inheritance 1 tlocs 1`' maps this inheritance pattern to the first trait locus. In this example, the trait locus follows marker 4 with a recombination fraction of 0.01, as is indicated by the statement '`map tlocs 1 marker 4 ...`'. The '`map marker positions`' statement specifies the spacing of the markers to be simulated, and also implicitly indicates the number of markers. The '`map marker positions`' statements beginning at line 4 must be included; they specify allele frequencies at the first two markers.

Following the '`set markers 10 data`' statement, the marker data availability is specified for each of the two associated alleles. A '0' indicates the data is unobserved, while a '1' indicates the data is observed. This specifies which alleles are to be output as data in the output simulated marker data.

The parameter file `ped73_mdrop_inhe.par` uses pedigree file `ped73.ped`. The file format section and first few lines of the pedigree data section of this file are below.

```
input pedigree record names 3 integers 7 reals 1
**************************************************
101 0 0 1 0 0 0 -1 -1 0 999.5
102 0 0 2 0 0 0 -1 -1 0 999.5
201 101 102 1 0 0 0 0 1 0 999.5
202 101 102 2 0 0 0 1 1 0 999.5
2010 0 0 2 0 0 0 -1 -1 0 999.5
301 201 2010 1 0 0 0 1 1 0 999.5
302 201 2010 2 1 3 2 1 1 0 105.945
```

The first three columns are indices of individuals and their parents. The next two are sex and observation status. Integer columns 5 and 6 are inheritance indicators with the first being the paternal ones and the second the maternal ones. A founder's meiosis indicators are '`-1 -1`'.

The connection to these inheritance data is through the statement

```
input pedigree record trait 3 integer pair 5 6
```

in the parameter file. Recall that on counting the pedigree file columns the integers follow the three names, so that integers 5 and 6 are columns 8 and 9 overall.

Note that `markerdrop` can only simulate data for markers linked to exactly one trait locus, as specified in the '`map`' statement in the parameter file.

For more information on `markerdrop` options see Section 6.5 [markerdrop statements], page 43.

See [Concept Index], page 151, for: `markerdrop` parameter file – conditional on inheritance pattern.

## 6.4 Running `markerdrop` examples and sample output

The `markerdrop` examples can be run while in the `Simulation/` subdirectory. The syntax for running a MORGAN program is:

```
<./program> <parameter file> [> <output file>]
or
<program> <parameter file> [> <output file>]
```

if your PATH includes your current directory.

Note that if the output file command is not included, the results will print to the console. To run a simulation of marker data conditional on a trait model, type the following into the console:

```
./markerdrop ped73_mdrop_trail.par > mdrop_trait.out
```

Likewise to simulate marker data conditional on an inheritance pattern, type the following:

```
./markerdrop ped73_mdrop_inhe.par > mdrop_inhe.out
```

After running `markerdrop` with the parameter file `mdrop_inhe.par`, and the pedigree file `ped73.ped` (as in the above example), the output file `mdrop_inhe.out` is generated. Some sections of this output file are given below. Note that similar output would be generated using `ped73_mdrop_trait.par`.

```
Inter-locus distances in cM, using Haldane map function:

                           T1
 ----------------------------+------------------------------------------
   10.0   10.0   10.0    1.0    9.0   10.0   10.0   10.0   10.0   10.0
 +------+------+------+------------+------+------+------+------+------+
M1     M2     M3     M4           M5     M6     M7     M8     M9     M10

 ......

 Assigned FGL in all listed individuals:
 trait locus, followed by 10 marker loci
 101  2 1  2 1  2 1  2 1  2 1  2 1  2 1  2 1  2 1  2 1  2 1
 102  4 3  4 3  4 3  4 3  4 3  4 3  4 3  4 3  4 3  4 3  4 3
 201  2 3  2 3  2 3  2 3  2 3  2 3  2 3  2 3  2 3  2 3  2 3
 202  2 4  2 3  2 3  2 3  2 4  2 4  2 4  2 4  2 4  2 4  2 4
 2010  6 5  6 5  6 5  6 5  6 5  6 5  6 5  6 5  6 5  6 5  6 5
 301  2 6  2 5  2 6  2 6  2 6  2 6  2 5  2 5  2 5  2 5  2 5
 302  2 6  3 6  2 6  2 6  2 6  2 5  2 5  2 5  2 6  2 6  2 6
 ......

 Assigned marker genotypes in accordance with data availability:
 101  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
 102  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
 201  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
 202  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
 2010  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
 301  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
 302  4 2  4 3  5 6  3 3  6 1  1 2  3 3  1 4  1 6  1 4
 ......
```

In the output file above, the marker map is shown, as specified in the parameter file. Below the map, founder genome labels (FGL) are listed. In this section of the pedigree, individuals 101, 102 and 2020 are founders and so each of them has been assigned two unique FGL. One of each founder's FGL has been randomly selected to be passed to their offspring. Using the FGL, marker genotypes have been assigned to individuals on whom data were specified as available in the parameter file, individual 302 for example.

The FGL and genotypes output by `markerdrop` are ordered (or phased). That is at each locus the paternal allele precedes the maternal allele. When these marker data are read by another MORGAN program they are read as unphased.

Also note that the `printlevel` has been set to 5 in this example; without doing so, the default behavior would be to omit printing the marker map as well as the FGL data.

See [Concept Index], page 151, for: running `markerdrop` examples, `markerdrop` output, founder genome labels, phased and unphased genotypes.

## 6.5 `markerdrop` statements

See [Concept Index], page 151, for: `markerdrop` statements.

### 6.5.1 `markerdrop` computing requests

`markerdrop` always requires the following statement:

`simulate [chromosome` *`I`*`] [dense] markers`

> This statement requests that markers are to be simulated. Whether the simulation is conditional on a trait model or on an inheritance pattern is inferred from the following statements. If the option to use dense markers is selected, descent is simulated by creating the recombination breakpoints in a meiosis, rather than simulating inheritance from marker to marker using recombination fractions.

`markerdrop` always requires one of the following two statements to establish whether the trait option or inheritance option is to be used.

`select trait` *`K`*

> This statement requests the simulation of markers conditional on a trait model using trait *K*. If marker data are simulated conditional on a trait model, the user must specify trait allele frequencies, genotypic penetrances and a map position for the trait locus within the parameter file. Affection status of each individual must be specified in the pedigree file following gender, if present.

`select inheritance` *`H`*

> This statement requests the simulation of markers conditional on an inheritance pattern at the trait locus. If marker data are to be simulated conditional inheritance pattern, the user must specify a map position for the trait locus within the parameter file. In addition, a pair of meiosis indicators for each individual must be included in the pedigree file following gender, if present. The first of the pair describes paternal inheritance at the trait locus and the second describes maternal inheritance. Inheritance indicators are coded as '`0`', '`1`' or '`-1`', corresponding to segregation of the trait allele from the individual's grandmother, grandfather, or unknown, respectively. For example, '`0 0`' indicates that the individual inherited the alleles carried by both grandmothers at the trait locus, while '`0 1`' indicates inheritance of the paternal grandmother's and maternal grandfather's alleles.

`set traits` *K1...* `tlocs` *L1...*
: This statement establishes the correspondence of traits to trait loci; it is used when the trait option is selected.

`set inheritance` *H1...* `tlocs` *L1...*
: This statement establishes the correspondence between loci and sets of partial inheritance indicators; it is used for the inheritance option. there may be more than one set of inheritance indicators assigned to a specific trait locus.

See [Concept Index], page 151, for: `markerdrop` computing requests marker simulation, marker simulation using trait, marker simulation using meiosis indicators.

### 6.5.2 `markerdrop` mapping model parameters

`map [gender (F | M)] marker ( [Kosambi] distances | recombination fractions | [Kosambi] positions)` *X1 X2* `...`
: This statement is required for `markerdrop` if more than one marker is to be simulated. It specifies the marker map (optionally a sex-specific map), in units of genetic distance (cM), marker position (cM), or recombination fraction. If distance is selected, `markerdrop` will expect one fewer values than the number of markers, as these are intermarker distances. If position is expected, the same number of values as markers will be expected, as these are the positions of the markers relative to some zero point to the left of marker 1. If Kosambi is not specified, the Haldane mapping function is used to convert between genetic distance and recombination fraction.

`map [gender (F | M)] tlocs` *K* `marker` *J* `( [Kosambi] distance | recombination fraction )` *X*
: This statement is required for `markerdrop`; it tells the program which trait locus to use in the simulation of marker data and gives a location for the trait locus, either as a map distance or recombination fraction, following the marker listed in the statement. As with `genedrop`, to simulate a trait locus position that precedes all markers, list the marker number as '`0`'.

See [Concept Index], page 151, for: `markerdrop` mapping model parameters

### 6.5.3 markerdrop population model parameters

`set tlocs` *K1* `allele frequencies` *X1 X2*
: This statement specifies trait locus allele frequencies. Trait loci must have two alleles; both allele frequencies must be listed and must sum to a value between 0.9999 and 1.0001. Otherwise `markerdrop` automatically normalizes the allele frequencies and issues a warning. Only one trait may be included in this statement.

`set [chromosome` *I*`] marker names` *N1 N2...*
: This statement specifies marker names in the order of their position along the chromosome. Default names are marker-1, marker-2, etc.

`set [chromosome` *I*`] markers` *K1* `... allele frequencies` *X1 X2* `...`
: Marker allele frequencies are specified using this statement. A marker can have up to 100 alleles and all allele frequencies must be listed. For each marker, the

allele frequencies should sum to between 0.9999 and 1.0001. Otherwise they are automatically normalized and a warning message will be issued. Multiple markers can be included in a single statement if they have the same number of alleles with the same frequencies.

See [Concept Index], page 151, for: `markerdrop` population model parameters, trait allele frequencies, marker names, marker allele frequencies.

### 6.5.4 `markerdrop` computational parameters

`set traits` *`K1`* `...` `for tlocs` *`L1`* `...` `incomplete penetrances` *`X1 X2 X3`*

This statement is required for `markerdrop` when using a trait model or when using meiosis indicators with a discrete trait. A penetrance, the probability of expressing the trait given a particular trait locus genotype, must be specified for each of the 3 possible genotypes at the trait locus. For example '`incomplete penetrances 0.15 0.85 0.99`' specifies that the probability of expressing the trait is 0.15, 0.85 and 0.99 for (1,1), (1,2) and (2,2) trait locus genotypes, respectively.

`set trait` *`K`* `data discrete`

This statement is optional. A discrete trait is the default when simulating conditional on trait data.

As with `genedrop`, marker seeds and trait seeds can be specified or the default values can be used, See Section 5.4.4 [genedrop computational parameters], page 35.

See [Concept Index], page 151, for: `markerdrop` computational parameters, penetrance, incomplete penetrance, discrete trait, trait data, marker seeds, trait seeds.

### 6.5.5 `markerdrop` input file options

The statements below are optional for `markerdrop`; they are used to indicate a change from the default order of trait values in the pedigree file. The first statement may be included if marker data are to be simulated conditional on a trait model and the second may be included if data are to be simulated conditional on an inheritance pattern.

`input pedigree record traits` *`K1 K2 K3`* `...` `integers` *`I1 I2 I3`* `...`

Unless this statement is present, the first integer following gender, if present, is assumed to be data for trait 1, the next integer for trait 2, and so on. Use this statement to specify an alternate correspondence between integer values in the record and trait numbers.

`input pedigree record inheritance` *`K1 K2`* `...` `integer pairs` *`I11 I12 I21 I22`* `...`

Unless this statement is present, the first two integers following gender, if present, in the pedigree file are assumed to be the meiosis indicators at the locus for trait 1. The next two integers are assumed to be the inheritance indicators at the locus for trait 2, and so on.

See [Concept Index], page 151, for: `markerdrop` input file options, pedigree record format.

# 7 Estimating *a priori ibd* Probabilities by Monte Carlo

See [Concept Index], page 151, for: *a priori ibd* probabilities, identity by descent, *ibd*.

## 7.1 Introduction to `ibddrop`

`ibddrop` estimates probabilities of gene identity by descent, *ibd*, (such as kinship, inbreeding, or multi-gene identities) by Monte Carlo in the absence of data. Given the pedigree and a genetic map, `ibddrop` simulates meioses indicators and scores them to estimate the *ibd* probabilities among a set of gametes. As originally written, the parameter format of `ibddrop` was set up to parallel that of the program `lm_auto` See Chapter 9 [Estimating Conditional IBD Probabilities by MCMC], page 63. The `lm_auto` program also estimates *ibd* probabilities, but does so conditionally on marker and (if requested) trait data. This format has been retained in the current `ibddrop` but it is important to recognize that in `ibddrop` 'markers' and 'tloc' (trait locus) refers only to a location on the chromosome, and not to any allelic or phenotypic entities.

The simplest example of estimation of *ibd* probabilities among a set of gametes is the computation of an individual's inbreeding coefficient. In this example, the set of gametes in question are the maternal and paternal gametes that make up the individual. A set of two gametes can be either *ibd* or not-*ibd*. To keep track of *ibd* status among the gametes, we can label the paternal allele '`1`'. If the two alleles are *ibd*, the maternal allele would also be labeled '`1`', and the resulting *ibd* pattern would be '`1 1`'. If the two alleles are not *ibd*, the maternal allele would be labeled '`2`' and the resulting pattern would be '`1 2`'. The individual's inbreeding coefficient is the probability that the two alleles follow the '`1 1`' pattern.

If there are three gametes in the set, there are five potential *ibd* patterns: '`1 1 1`' (all three gametes are *ibd*), '`1 1 2`' (the first two are *ibd* and the third is not), '`1 2 1`' (the first and third are *ibd*) , '`1 2 2`' (the last two are *ibd*), and '`1 2 3`' (none are *ibd*). `ibddrop` can estimate probabilities of *ibd* patterns among up to 10 gametes in a set. `ibddrop` outputs a probability for each *ibd* pattern at each marker.

Gene identity can be scored either for each locus separately, in which patterns of identity among up to ten gametes can be scored, or it can be scored jointly over a moving window of several loci. If the moving window option is selected, `ibddrop` estimates the probabilities of each *ibd*/non-*ibd* pattern at loci across the window, for the specified pair of gametes.

For MORGAN V3.4, the `ibddrop` program has been extensively revised.

- Descent is simulated at specific 'marker' locations. Inclusion of a trait locus position is optional. If included it may be either linked or unlinked to the marker msp. IBD may be scored either at each locus, or an IBD patters over a window of several markers may be scored.
- For clarity, 'MC' in MORGAN now refers to MCMC realizations only. The parameter statement 'set MC iterations' in `ibddrop` has been change to 'simulate ibd realizations'.
- A "dense" option was added to the "simulate ... markers" parameter statement. Using this option causes simulation of meioses, with recombination breakpoints occurring in a continuum down to 10^-6 centiMorgans (effectively, one base pair). The scoring of IBD patterns over loci and outputting of IBD probabilities remain the same as before.

The capability of scoring jointly over a moving window is not currently available for the "dense" option.

- Optionally, implemented as a beta-test option, selection resulting in segregation distortion can be imposed at the trait locus, when this is specified as the left-most locus. The parameters of this genetic selection process are entered using the 'dummy reals' statement See Section 2.5 [Input extra variables], page 13.
- An additional beta-test option is implemented post-MORGAN 3.4 release. In this option kinships and inbreeding coefficients only are output to an *output scores file*. The 'dummy integers' statement input the numbers of individuals in two lists 'P' and 'D'. The pedigree indices of individuals in the two lists are read from the *input extra file*. Output are the estimated inbreeding coefficients of the individuals in both lists, and the kinship of each individual in list-D with each in list-P.

See [Concept Index], page 151, for: `ibddrop` introduction, *ibd* pattern, meiosis indicators, simulation of descent in a pedigree.

## 7.2 Sample ibddrop parameter file

See the [Concept Index], page 151, for: `ibddrop` sample parameter file,

### 7.2.1 Classic ibddrop parameter file

The example parameter file `jv_ped_ibd.par` in the `IBD` subdirectory of `MORGAN_Examples` has been updated so that it will run under MORGAN V3.4. So also have examples parameter files `ibddrop1.par` and `ibddrop2.par`. Details may be found in `README_ibd` in that same subdirectory. However, for convenience, we describe here examples in the `Gold` subdirectory of the `Genedrop` program directory. One such example is the parameter file `parIBD_LL`:

```
set printlevel 5

input pedigree file "ped45"

simulate markers
simulate tloc 11

map          markers   recomb fract .18 .1 .1 .1
map tloc 11  marker 2  recomb fract .06

set component 1  scoreset 1  proband gametes 331 0 333 1
set component 1  scoreset 2  proband gametes 531 0 531 1 331 0 333 1
set component 2  scoreset 1  proband gametes 3v1 0 3v3 1
set component 2  scoreset 2  proband gametes 5v1 0 5v1 1 3v1 0 3v3 1
set component 3  scoreset 2  proband gametes 5w1 0 5w1 1 3w1 0 3w3 1
set component 3  scoreset 1  proband gametes 3w1 0 3w3 1

set sampler seeds 0x8a226a51 0xd2978c71

simulate 20000 ibd realizations
```

The parameter file specifies the pedigree file name `ped45` and then asks simulation at markers and at one trait locus. The number of markers is determined by the map statement: since there are 4 recombination parameters provided, there will be 5 marker locations in addition to the trait location. Note that, since there are no data, this is simply a way to specify 6 locations, one of which (the `tloc`) may play a special role and/or may be unlinked. `ped45` contains 45 individuals, who are 3 replicates of the JV pedigree. The file includes also gender and a 'trait' but trait data are ignored by `ibddrop`. (Other programs which use the trait data, such as `lm_auto` may also used this same pedigree file. See Chapter 9 [Estimating Conditional IBD Probabilities by MCMC], page 63.)

The two '`map`' statements specify the genetic map. From the first statement, the genetic distances between the markers are 44.6, 44.6, 11.2 and 11.2 centiMorgans. From the second statement, the trait lies between markers 2 and 3, at 22.3 centiMorgans with marker 2.

The '`set proband gametes`' statements tell `ibddrop` which gametes to score: that is, the gametes among which the *ibd* probabilities will be estimated. In this example, we selected, from component 1 (the first family in the data set), the maternal (0) gamete of '`331`' and the paternal (1) gamete of '`333`'. The next statement selected four gametes to score from family 2. Note that characters are allowed in the names of individuals.

The '`input seed file`' statement enables the file to use the seeds from file `sampler.seed`. The '`output overwrite seed file`' statement allows the program to replace the contents of the seed file with the newly generated seeds. If this options were omitted, when the program finished running, new seeds would be appended to the end of the file. Seeds can also be set using the '`set sampler seeds`' statement (see Section 7.4 [ibddrop statements], page 54).

The number of Monte Carlo realizations is set to be 20,000 by the '`simulate ibd realizations`' statement.

Note that to compute a multilocus *ibd* probability, the statement '`set locus window`' can be used to specify the number of loci to score jointly. `ibddrop` has limited functionality for computing multilocus probabilities, it can only examine two gametes to determine whether or not the two are *ibd*. For an example see the parameter file `parIBD_WL` where the statement '`set locus window 3`' is included, and each proband gamete set is of two gametes only. For additional options, including scoring of a specific multi-gamete *ibd* pattern over two or more loci, see the file Section 9.2 [Sample lm_auto parameter file], page 64: `lm_auto` has this more general option of scoring multi-gamete *ibd* patterns.

See [Concept Index], page 151, for: map function, proband gametes, seeds for sampler, seed file.

### 7.2.2 Using dense marker simulation

An example of the 'dense' simulation is given in the '`Gold`' file dense_markers.par:

```
set printlevel 5

map chromosome 2 markers  recomb fract .1 .05 .15 .2
map chromosome 3 markers  recomb fract .04 .12 .1 .1
map chromosome 4 markers  recomb fract .18 .1 .1 .1

set component 1  scoreset 1  proband gametes 331 0 333 1
set component 1  scoreset 2  proband gametes 531 0 531 1 331 0 333 1
set component 2  scoreset 1  proband gametes 3v1 0 3v3 1
set component 2  scoreset 2  proband gametes 5v1 0 5v1 1 3v1 0 3v3 1
set component 3  scoreset 1  proband gametes 3w1 0 3w3 1
set component 3  scoreset 2  proband gametes 5w1 0 5w1 1 3w1 0 3w3 1

simulate chromosome 4 dense markers

simulate 100 ibd realizations

# Tell ibddrop where to read the pedigree from.
input pedigree file           "./ped45"

# Provide a file name for the ibddrop ibdgraphs results file.
output overwrite scores file  "./dense_markers.ibdgraphs"

# Set the sampler seeds.
set sampler seeds 0x8a226a51 0xd2978c71

# The following select markers parameter statement tells ibddrop
# where to score and output IBD probabilities at.

select chromosome 4 markers 1 3 5
```

Here the pedigree '`ped45`' and proband gametes are as before. Locations on 3 different chromosomes are given, but just one (here '`Chromosome 4`' must be chosen for simulation of *ibd.* the '`simulate dense markers`' option specifies that simulation will be by generating recombination breakpoints in a continuum, rather than using marker-to-marker computations, and again a number of '`ibd realizations`' is requested. An '`output scores file`' is optionally provided. If it is, the simulated '`ibdgraphs`' will be output in compact format. See Section 4.6 [The ibd_class utility], page 26. The scoring of IBD patterns over loci and outputting of IBD probabilities remain the same as before. Optionally, the '`select markers`' statement can be used to specify a subset of the original 'marker' locations at which to score *ibd.* If this statement is not present, *ibd* will be scored at all the mapped locations for that chromosome. The length of the simulated chromosome is determined by the distance between the first and the last selected markers.

Although here in this small example only a few locations are specified, this 'dense' option is intended for cases where there are many marker locations, so marker-to-marker simulation is inefficient. If the '`ibdgraph`' is output, subsets of marker locations for later scoring may later be selected without re-simulation. See Section 12.6 [Sample parameter files for ibd_create; fgl2ibd and fgl2haplo and fgl2dgl], page 123. Since scoring is done at each specified location, this part of the computation will be linear in the number of markers selected. With very large numbers of selected markers, it is suggested that '`printlevel 3`' is a better option than '`printlevel 5`' to avoid attempts to print the marker map. The capability of scoring jointly over a moving window is not currently available for the "dense" option.

See [Concept Index], page 151, for: dense markers

### 7.2.3 Selection against autozygosity

Examples in which selection is imposed at the first location (the '`tloc`') are provided in the two `Gold` files `ibd_cleo_sparse.par` and `ibd_cleo_dense.par`. An additional example, `cleo_segments.par`, is included in the `IBD` subdirectory of `MORGAN_Examples`. These examples are so named because they use the very highly inbred Cleopatra pedigree, `cleopatra.ped`.

```
# for running "ibddrop"  with selection against autozygosity and against
#    mom-identity, at target locus 0,  on Cleopatra pedigree
# Aug 6, 2017 -- new version for Gold standard
#    Note this version prints the FGL statistics both to file and to output
#    It also prints autozygosity to the extra output file
#

input pedigree file 'cleopatra.ped'

# Include everything in the output file.
set printlevel 5

simulate markers
simulate tloc 11

map          markers   distances  1 1 2 3 5 8 13 21 34 55
map tloc 11  marker 0  distance 1

set component 1  scoreset 1  proband gametes
Ptolemy-VIII  0 Ptolemy-VIII 1 Cleopatra-III  0 Cleopatra-III 1
set component 1  scoreset 2  proband gametes
Berenice-III  0 Berenice-III 1 Cleopatra-VII 0 Cleopatra-VII  1
set component 1  scoreset 4  proband gametes
Cleopatra-V 0 Cleopatra-V 1 Cleopatra-VII 0 Cleopatra-VII  1

set sampler seeds 0x8a226a51 0xd2978c71

simulate 100000 ibd realizations

output overwrite extra file "ibd_cleo_sparse.scor"

set dummy reals 0.9, 0.5, 0.9    # selection at 0.9 against autozygosity
                                 # selection of 0,5 against mom-identity
                                 #    and 0.9 for both
```

Much of the parameter file is as before, but note that the `tloc` at which selection is to be imposed should be at the left end of the set of markers at which *ibd* is to be scored. As before, *ibd* probabilities among the proband gametes is printed to the standard output. Since in `ibddrop`, the '`output scores file`' (if requested) is used to save the realized *ibd* graphs, the '`ioutput extra file`' is used to print the autozygosity statistics to file for possible later analyses.

Selection is imposed at the target `tloc` by means of three selection indices input via the '`set dummy reals`' statement. See Section 2.5 [Input extra variables], page 13. The three numbers (0.9, 0.5 and 0.9 in the above example) are viability weights relative to a norm of 1. The first is for a potentially autozygous offspring (*ibd* between the two offspring gametes), the second for a potential offspring being identical (*ibd* at both gametes) to the mother, and the third is for a potential offspring with both characteristic (so that all four gametes of offspring and mother are *ibd*). Selection at the '`tloc`' is imposed by modifying the segregation at this location. Descent at the linked markers is then simulated conditional on descent at the '`tloc`'. Note that selection does not change the pedigree structure, but only the segregation of DNA within the pedigree. The Cleopatra pedigree is chosen for the example, because its extreme inbreeding results in major effects of both types of selection.

The parameter file for dense marker simulation in this example is `ibd_cleo_dense.par`. It differs only in the statements:

```
simulate dense markers
output overwrite extra file "ibd_cleo_dense.scor"
```

Modulo variation due to randomness, the results for these two versions are identical. Only the methods of simulation of meiosis across the chromosome differs, with the '`sparse`' realizations being done marker to marker using recombination fractions, and the '`dense`' version generating recombination breakpoints.

An alternative is provided in the parameter file `cleo_segments.par`. This file was not included in the MORGAN V3.4 release, but instead is now added to `MORGAN_Examples/IBD`. This is very similar to the `cleo_dense.par` file, but differs in that the FGL graphs are additionally printed. The `output scores file` is used for these FGL graphs. From the FGL graphs, segments of *ibd* among any set of gametes can be reconstructed and scored, rather than scoring marker-by-marker.

See [Concept Index], page 151, for: Cleopatra pedigree, selection against autozygosity, selection against maternal identity.

## 7.3 Running `ibddrop` example and sample output

The syntax for running this MORGAN program is:

```
<./program> <parameter file> [ > <output file name> ]
```

where , optionally, '`>`' redirects the standard output (<stdout>) to an output file instead of to the screen.

For example, the `parIBD_LL` example can be run in the `Gold` subdirectory of `Genedrop` with the following command:

```
../ibddrop parIBD_LL  > ibddrop.out
```

The genetic map specified by the statements '`map markers recomb fract`' and '`map tlocs 11 marker 2 recomb fract`' is below. Note the position of the trait locus (*T11*) with respect to the marker loci.

```
 Chromosome map
 ..............
```

```
      Inter-locus distances in cM, using Haldane map function:

                   T11
      --------------+---------------------
        22.3    6.4    4.8   11.2   11.2
      +------+-------------+------+------+
     M1     M2            M3     M4     M5
```

Since the parameter file contains six'`set proband gametes`' statements, `ibddrop` will produce six sets of results in the output file (here '`ibddrop.out`').

The exact probability estimates will, of course, depend on the random seed used. Some example results for the second component are detailed below.

```
      Summary for component 2:

         Probabilities of IBD patterns

            Proband gamete set 1:  3v1 0  3v3 1

            pattern marker-1 marker-2  tloc-11 marker-3 marker-4 marker-5    label

               1 1    .2526    .2497    .2517    .2509    .2495    .2500        0
               1 2    .7473    .7503    .7483    .7491    .7506    .7500        1

         Probabilities of IBD patterns

            Proband gamete set 2:  5v1 0  5v1 1  3v1 0  3v3 1

            pattern marker-1 marker-2  tloc-11 marker-3 marker-4 marker-5    label

            1 1 1 1    .0314    .0300    .0309    .0303    .0290    .0294        0
            1 1 1 2    .0284    .0272    .0278    .0288    .0290    .0289        1
            1 1 2 1    .0149    .0144    .0135    .0129    .0137    .0143        3
            1 1 2 2    .0100    .0087    .0092    .0092    .0092    .0092        4
            1 1 2 3    .0263    .0301    .0283    .0277    .0271    .0263        5
            1 2 1 1    .0658    .0649    .0637    .0627    .0631    .0625        6
            1 2 1 2    .0056    .0051    .0060    .0060    .0056    .0063        7
            1 2 1 3    .0589    .0593    .0583    .0590    .0588    .0594        8
            1 2 2 1    .0648    .0678    .0697    .0701    .0714    .0698        9
            1 2 2 2    .0494    .0505    .0503    .0507    .0486    .0493       10
            1 2 2 3    .1366    .1416    .1389    .1386    .1393    .1387       11
            1 2 3 1    .1366    .1348    .1349    .1346    .1330    .1349       12
            1 2 3 2    .0296    .0279    .0265    .0251    .0260    .0280       13
            1 2 3 3    .0961    .0955    .0975    .0979    .0995    .0995       14
            1 2 3 4    .2455    .2420    .2444    .2464    .2469    .2434       15
```

The probabilities are summarized by the *ibd* pattern. Each integer in the pattern represents one of the gametes that `ibddrop` was asked to score. Same numbers indicate gametes that are *ibd*. For instance, '`1 1 1 1`' means all four gametes are *ibd*; '`1 2 1 1`' means gametes 1, 3, and 4 are *ibd*, while gamete 2 is not *ibd* with the others; '`1 2 3 4`' means all four gametes are not *ibd*.

The *ibd* patterns are scored for each locus separately; there is a column for each of the five markers and one for the trait locus. The final column 'label' is a label for the state that can be easily inverted to obtain the *ibd* pattern; its main use is internal to the program. However, the in `lm_auto` program one may request scoring of specific *ibd* patterns by specifying the desired state labels (see Section 9.2 [Sample lm_auto parameter file], page 64).

To compute multilocus *ibd* probabilities, say for 3 loci, use the parameter file `parIBD_WL` which contains the line '`set locus window 3`'. The interesting part of the output for component 2 is:

```
          Probabilities of IBD patterns for windows of 3 loci

             Proband gamete set 1:  5v1 0  5v1 1

               IBD  wndw 1 wndw 2 wndw 3 wndw 4

             0 0 0   .7826  .7680  .7902  .7919
             0 0 1   .0255  .0459  .0473  .0471
             0 1 0   .0712  .0377  .0264  .0249
             0 1 1   .0109  .0374  .0257  .0274
             1 0 0   .0312  .0694  .0488  .0484
             1 0 1   .0496  .0062  .0050  .0047
             1 1 0   .0045  .0161  .0267  .0268
             1 1 1   .0244  .0192  .0300  .0288
```

This time, `ibddrop` was asked to compute *ibd* probabilities in windows of three loci at a time. The four windows can be seen from the marker map: (M1,M2,T11), (M2,T11,M3), (T11,M3, M4), (M3, M4, M5). If the trait locus were unlinked to the marker loci, it would be placed to the left of the five marker loci on the map. Thus the first window, '`wndw 1`', would include the trait locus and the first two marker loci. The values in the '`ibd`' column at the left of the table represent '`ibd`' patterns. The pattern '`0 0 0`' means that the selected gametes are not *ibd* at the three loci in each window. The pattern '`0 0 1`' means that the selected gametes are not *ibd* at the first two loci in the window, but are *ibd* at the third. The values in the columns give the probability of the *ibd* pattern at the left for each of the four windows. For example, the probability that the maternal and paternal gametes of individual 5v1 are *ibd* at marker loci 3 and 5, but not at marker locus 4 is 0.0047.

The format of output for the dense marker options is effectively identical to the above: only the simulation of meioses and recombination breakpoints differ. Example outputs for the beta-test version of `ibddrop` with selection may be found in the `Gold` subdirectory of `Genedrop`. The relevant files all contain the string `cleo`, since the examples use the Cleopatra pedigree. The output for this beta-test program will not be further discussed here.

The alternative of outputting FGL graphs, and scoring *ibd* directly by segments as mentioned in see Section 7.2.3 [Selection against autozygosity], page 50, is provided in the parameter file `cleo_segments.par`. For reference this is included in the `IBD` subdirectory of `MORGAN_Examples`. Note that, for compatibility with MORGAN V3.4, it is also necessary to output minimal regular scoring output. However, the intention would be to use the simulated FGL graphs in any downstream analyses.

See [Concept Index], page 151, for: running `ibddrop` example, `ibddrop` sample output, *ibd* pattern.

## 7.4 `ibddrop` statements

- Use the '`simulate markers`' statements to specify simulation of markers and one linked or unlinked trait, for each of one or more chromosomes (see Section 5.4.1 [genedrop computing requests], page 33). For convenience these key statements are repeated below.
- Use '`map`' statements to specify the marker and trait locus maps (see Section 5.4.2 [genedrop mapping model parameters], page 33). Note also one additional '`map`' statement below.

Note that `ibddrop` does not simulate or use marker or trait data. The statements are used only to specify the map of the loci at at which descent is to be simulated and *ibd* scored. The locations of loci are specified in this way so that direct comparisons can be made between output of `ibddrop` and of `lm_auto` (see Section 9.3 [Running lm_auto example and sample output], page 67), where simulation is conditional on marker and trait data.

The additional `ibddrop` statements are:

`simulate [chromosome` *`I`*`] [dense] markers`

> This statement requests that markers are to be simulated. For '`ibddrop`' the name 'markers' refers only to chromosome locations, not to actual alleles or individual genotypes. The number of markers (i.e. locations) is inferred from the marker map. If the option to use dense markers is selected, descent is simulated by creating the recombination breakpoints in a meiosis, rather than simulating inheritance from marker to marker using recombination fractions.

`simulate tloc` *`L`*

> This statement, which typically follows the `simulate markers` statement, establishes the trait locus to be simulated. Note that this trait locus must be mapped onto the chromosome selected for marker simulation.

`map tlocs` *`L1`* `... unlinked`

> This statement specifies a trait to be simulated that is not linked to markers. Only one trait can be simulated and this trait will be placed to the left of all markers.

`set [component` *`M`*`] proband gametes` *`N1 K1 N2 K2...`*

> In this statement, the user specifies which gametes `ibddrop` is to score. Each statement must contain gametes from a single component, as the components are assumed to be independent, i.e. the probability of *ibd* between gametes from different components is zero. Pairs consisting of an individual's name and a meiosis indicator are listed, with '`0`' indicating the individual's maternal gamete and '`1`' indicating their paternal gamete.

In the current version of MORGAN, the number of proband gametes in a set is limited to 10.

`set [chromosome` *I*`] locus window` *K*

This statement gives the window size (number of loci) for which the multilocus *ibd* probabilities are scored. If no size is given, each locus is scored separately.

`set sampler seeds` *H1 H2*

This statement initializes a pair of seeds for the random number generator. The seeds must be positive and no greater than '`0xFFFFFFFF`', with the first seed (congruential seed) odd, and the second seed (Tausworthe seed) nonzero. If no seeds are specified, default seeds are used.

`simulate` *K* `ibd realizations`

This statement is requited for `ibddrop`.

See [Concept Index], page 151, for: `ibddrop` statements, proband gametes, meiosis indicators, seeds for sampler.

# 8 Using MCMC to Estimate Parameters of Interest in Pedigree Data

See [Concept Index], page 151, for: MCMC introduction, Markov chain Monte Carlo.

## 8.1 Specifying inheritance

See [References], page 149, for details of the cited papers.

For MORGAN programs, genetic relationships between individuals in a data set are specified in the pedigree file. Individuals at the top of a *pedigree* (family), whose parents are unspecified, are the *founders* of the pedigree; other individuals are *non-founders*. In pedigrees, identity by descent is defined relative to the founders of the pedigree, so that, by definition, founders are unrelated to one another. Descent through the pedigree of genes at marker and trait loci is tracked by *meiosis indicators*, also known as *inheritance indicators* or *segregation indicators* [Don83]. At each locus, non-founders are assigned two `0/1` meiosis indicators, representing genes inherited from the individual's father and mother. The first indicator is coded as '`0`' if the non-founder inherited the gene carried by her father's mother and '`1`' if she inherited the gene carried by her father's father, i.e. her paternal grandmother and grandfather, respectively. The second indicator is coded as '`0`' if the non-founder inherited the gene carried by her mother's mother and '`1`' if she inherited the gene carried by her mother's father, i.e., her maternal grandmother and grandfather, respectively. We use the term *gene* to refer to a segment of DNA that is copied from parents to offspring, the concept captured by Mendel's term *factor*.

The set of all meiosis indicators is denoted **S** = ( *Sij* ) where

$$Sij = \begin{cases} 0 \text{ if DNA involved in meiosis } i \text{ at locus } j \text{ is the gamete's parent's maternal DNA} \\ 1 \text{ if DNA involved in meiosis } i \text{ at locus } j \text{ is the gamete's parent's paternal DNA} \end{cases}$$

The vector of meiosis indicators at a single locus *j* over all the meioses of a pedigree is known as the *inheritance vector* at locus *j* [LG87] and is denoted *S.j*. The elements of *S.j* are independent of one another, as they represent the inheritance to gametes resulting from different (and hence independent) meioses. *Si.* is the vector of meiosis indicators at all loci for a single meiosis *i* (that is, in the formation of a single gamete). Assuming the absence of genetic interference [Hal19], the elements of *Si.* have first–order Markov dependence. That is, the value '`0`' or '`1`' at locus *j* `+ 1`, given the values at loci *1, 2, ...., j* depends only on the value at locus *j*. Specifically, this probability is a function of the value at locus *j* and the recombination fraction between the loci *j* and *j*`+1`.

If meiosis indicators are known, identity by descent (*ibd*) is also known. If probabilities can be assigned to patterns of meiosis indicators in a pedigree, the probability that any set of gametes in the pedigree are *ibd* can in principle be computed.

See [Concept Index], page 151, for: specifying inheritance, meiosis indicators, founder genome labels, inheritance vector, *ibd*.

## 8.2 Genetic model

See [References], page 149, for details of the cited papers.

In MORGAN there are three basic genetic data types. These are: be *genotypic*, typically used for marker data; *discrete*, a data type for binary data requiring specification of *incomplete* *penetrances*; and *quantitative*, using a Gaussian penetrance with specification of genotypic means and residual variance.

As yet, loci are either multiallelic marker loci assumed observed without error, or trait loci which may have general penetrance functions but must have two alleles. Gradually, available models are being generalized:

1. Pedigree peeling for multiallelic loci with general penetrance;

   In order to allow models for "non-genotypic" markers, general joint peeling programs have been implemented, based on Thompson in [Tho77]. For zero-loop pedigrees (see [Tho76]), these peeling routines are used by the `lm_map` program which allows for errors in marker data. For general pedigrees, they are not yet released, as they are still in process of testing.

2. Penetrance functions and trait models:

   Liability classes been implemented for the discrete-trait penetrance model. A parameter statement specifies the number of liability classes and the 3 genotype-specific penetrances in each class. Additionally, an age-based penetrance function for a qualitative trait has been implemented, with age specified as a covariate for the discrete trait. The mean ages of onset for each genotype are provides via the `genotypic means` parameter statement, and standard deviations are separately provides. An additional parameter statement specifies a window-width for the age-of-onset data.

3. Traits and trait loci:

   The program `lm_twoqtl` allows two (linked or unlinked) quantitative trait loci to contribute additively or epistatically to a single trait [STW07]. A polygenic component may also be also be included. Two-locus penetrances may be specified as additive, with a genotypic mean for each trait genotype for each locus. Alternatively, a matrix array of 2-locus genotypic means may be specified, allowing for epistasis [SW07].

With more these more complex trait models, including those of `lm_twoqtl` [STW07], a more general specification of traits is required. From MORGAN V3.0, completely new structures have been introduced, separating traits (phenotypes) from trait loci ("tlocs"). Traits may be affected by genotypes at several tlocs; the genotypes at a tloc may affect several traits.

See [Concept Index], page 151, for: genetic model, penetrance.

## 8.3 Exact HMM computations

Using the inheritance vectors or meiosis indicators, the structure of the problem is that of a hidden Markov model (HMM) with the Markov latent state being the *S.j*, Markov over markers *j*. When the pedigree is small, so that each *S.j* takes only a practical number of values, standard exact HMM computational methods apply. Likelihoods and lod scores can be computed exactly. Alternatively, a single forwards computation followed by (repeated) backwards sampling provides multiple independent realizations from the joint distribution of all the *Sij* given the marker data (or given the marker and trait data, if the latter is included in the set of loci *j*).

Note that in fact *Sij* are independent over meioses *i*, so that the structure is that of a factored HMM. Forward HMM computation for multiple meioses has been replaced by a factored version (FHMM), enabling much faster exact computation on small pedigree components and multiple-meiosis sampling for larger numbers of meioses.

Exact computation of lodscores on small pedigree components has been implemented for `lm_linkage` using the FHMM version of the Baum algorithm. Additionally, these FHMM computations are also a component of MCMC sampling on larger pedigree components (see next section).

Gold standards for exact computation are added in the Lodscore/Gold2 subdirectory.

See [Concept Index], page 151, for: exact computations, HMM computations.

## 8.4 Single and multiple meiosis LM-samplers

See [References], page 149, for details of the cited papers.

MORGAN's Autozyg and Lodscore programs use MCMC to estimate *ibd* probabilities and multilocus lod scores, respectively, in pedigrees. The latent (unobserved) parameters of interest in MCMC estimation of *ibd* probabilities and lod scores are the meiosis indicators at marker and/or trait loci for each non-founder in the pedigree. Observed data are trait values and *unphased* marker genotypes for some or all pedigree members. With unphased genotypes, it may or may not be possible to determine the grandparental source (i.e. the meiosis indicator) of each allele unambiguously. MORGAN uses MCMC to sample meiosis indicators (**S**) conditional on observed data (**Y**).

MORGAN implements two different block Gibbs samplers, a locus- and a meiosis-sampler, for sampling from **S** conditional on **Y**. Each method updates a subset, *S_u*, of **S** conditional on **Y** and on the currently fixed values of the rest of **S** (*S_f*). The difference between the two methods is the choice of *S_u*.

The locus-sampler (or L-sampler) chooses *S_u* to be *S.j* for some *j*. In other words, a single locus is selected and meiosis indicators at that locus are updated based on the genotype data at all loci and on the current realization of meiosis indicators at all loci other than *j*. The MORGAN user can determine whether a locus is to be selected at random each time or if loci are taken in a pre-determined random order, as described in the next section. The update computations use a modification of the Elston-Stewart algorithm [ES71] and can be used whenever single locus pedigree peeling is possible. If inter-locus recombination fractions are strictly positive, the L-sampler is irreducible. On the downside, mixing is poor if loci are tightly linked.

The single-meiosis sampler (or M-sampler) chooses *S_u* to be *Si.* for some *i*. It is, in a sense, perpendicular to the L-sampler in that at each iteration a single meiosis is selected and meiosis indicators for that meiosis are updated conditional on the genotype data at all loci and the current realization of meiosis indicators for all other meioses. The M-sampler is a modification of the Lander-Green algorithm [LG87] for peeling along a chromosome using the Baum algorithm [Bau72]. At each iteration, a single meiosis is randomly selected or meioses can be updated sequentially. As with locus selection in the L-sampler, MORGAN allows the user to choose the meiosis selection The M-sampler mixes well in the presence of tightly linked loci, but it can perform poorly in large pedigrees with missing data.

The multiple meiosis sampler updates several meioses jointly and is therefore a generalization of the old single-meiosis sampler. There are four types of multiple-meiosis updates: random meiosis update, individual update, sib update and 3-generation update. This is based on work by Liping Tong in [TT08]. The new LM-sampler is a combination of L-sampler and multiple-meiosis M-sampler. This new LM-sampler is implemented in the program `lm_linkage`, which combines the earlier programs `lm_markers` and `lm_multiple`.

The new LM-sampler can also be used in the programs `lm_auto` and `gl_auto`, and in the program `lm_twoqtl`. All these MORGAN 3.0 programs sample inheritance patterns conditional on marker data for use in subsequent lod score or *ibd* computations. They all have the option to use either the old (single-meiosis) or new (multiple-meiosis) LM-sampler.

MORGAN's Autozyg and Lodscore programs use a combination of the L- and M-samplers, referred to as the LM-sampler. The user may choose the fraction of updates that are of each type; the default is 20% L-sampler, 80% M-sampler. The recommendation is 20/80, 50/50 or 80/20, depending on which sampler is, in any particular example, the more computationally intensive.

For original descriptions of the L-sampler see [He97], and for the M-sampler and LM-sampler see [TH99]. For additional mathematical details on the L-, M- and single-meiosis LM-samplers, see [Tho00]. For the new multiple-meiosis sampler see [TT08].

Up to MORGAN V2.8.2, MCMC was performed globally over pedigree components (except those small enough for exact computation). The L-sampler peeling and lod score estimation could be done either by component (using "set peeling by component") or globally (the default).

With MORGAN V3.0, the preferred option is to do both MCMC and pedigree peeling (lod score estimation) by component, and to use exact computation on all sufficiently small component pedigrees. The alternative, retained so that older data sets can be rerun, is to use '`set global MCMC`', in which case no exact computation will be done, and MCMC will be done globally over all component pedigrees.

The `lm_haplotype` program is a generalization of `lm_multiple` in which haplotypes of key individuals dividing the pedigree are sampled in addition to meiosis indicators. To facilitate efficient implementation of this algorithm, new peeling-by-component routines need to be implemented and checked. This program is also the work of Liping Tong. This program is not yet released.

See [Concept Index], page 151, for: LM-sampler, pedigree peeling, multiple meiosis sampler, block Gibbs sampler, L-sampler, M-sampler, L-sampler probability.

## 8.5 MCMC computational options

MORGAN can obtain a starting configuration for **S** in one of two ways. The default method is by sequential imputation. The alternative is to construct an L-sampler realization independently for each locus, conditional on the genotype data at that locus only (the locus-by-locus option). Sequential imputation tends to produce initial configurations that have higher conditional probabilities, but locus-by-locus sampling can sometimes reveal other modes in the complex space of **S** values. The MORGAN user can select the independent-loci setup method by including the '`use locus-by-locus for setup`' statement. If sequential imputation is selected, the user can specify the number of sequential imputation samples from which the starting configuration of meiosis indicators is to be selected, using the '`use` *I* `sequential imputation realizations for setup`' statement. The default is 10% of the total MC iterations.

At each MCMC iteration, MORGAN selects a locus (with L-sampler) or set of meioses (with M-sampler) to update. Two different selection methods are available: sample by step and sample by scan. If '`sample by scan`' is chosen, all loci or meioses are updated one-at-a-time in a predetermined random order. This option is the default. If '`sample by step`' is chosen, a single locus or meiosis is randomly selected for updating at each iteration. The sampling method selected applies to the entire MCMC run, including burn-in, pseudo-prior computation and main iterations.

When running a MORGAN MCMC program, the user must specify the desired number of several types of iterations. For all programs, some number of initial burn-in iterations must be performed. These realizations are discarded and, if the burn-in period is sufficiently long, subsequent points will be dependent samples from approximately the desired stationary distribution. The '`set burn-in iterations`' statement is used to specify the number of desired burn-in iterations, with the default value varying by program. The desired number of '`main`' iterations must be specified using the '`set MC iterations`' statement; there is no default number of main iterations. For real-data analyses the recommended number of iterations is on the order of 10^5.

The `lm_bayes` program samples not only meiosis indicators, but also the location of the trait locus. This is done via a third type of MCMC Metropolis-Hastings update. The counts of sampled trait-locus locations is used to calculate pseudo-priors, which are then used in lod score estimation. Alternatively, pseudo-priors can be read from an input file. The goal of this two-stage procedure is to weight locations in order to encourage the MC sampler to visit test positions of low conditional probability. The number of iterations for calculation of pseudo-priors is set using the '`set pseudo-prior iterations`' statement, or the default value of 50% of the number of main iterations can be used.

Specific Autozyg and Lodscore programs have additional parameters and options that are described in the relevant sections of the next three chapters of the tutorial.

In addition to the main program-specific outputs described in the following chapters, the MCMC process accumulates diagnostic counts, scoring the configuration of meiosis indicators at intervals determined by the same statement compute scores every I iterations as is used for scoring for the primary output. (By default, scores and diagnostic output are computed every iteration.)

There are three components to the MCMC diagnostic output:

1. Average total log-probability of segregations:

   This is the average (over the scored iterations) of the total (over meioses) of the log-probability of the meiosis indicators. For the first locus this is simply the marginal probability log((1/2)^m) for m meioses, and for each successive locus is log P(*S.j* | *S.(j-1)*) for locus *j* conditional on locus *j-1*.

2. Average total log-probability of penetrances, by locus

   This is the average (over the scored iterations) of the combined (over observed individuals) log-probability of the observed data at the locus given the inheritance configuration *S.j*.

3. Recombination counts for map intervals

   This is the total count over (male and female) meioses and over MCMC iterations of realizations of configurations of meiosis indicators that are recombinant and non-recombinant in each interval of the map.

In these diagnostic scores, for the programs `lm_pval` and `lm_linkage` only marker loci and marker map intervals are included in these diagnostic scores. For `lm_auto`, the trait locus (designated '0') is included in the correct position, if it is included in the MCMC, while the `gl_auto` program requires a null (no-data) unlinked trait locus. This null locus may or may not be included in the MCMC sampling: see '`set MCMC markers only`' in Section 9.8.8 [Autozyg MCMC parameters and options], page 79.

See [Concept Index], page 151, for: MCMC options, sequential imputation, locus-by-locus setup, sample by step, sample by scan, burn-in, MC iterations, pseudo-prior iterations.

## 8.6 MCMC parameter statements

These statements which set the parameters for the MCMC algorithms, apply to both Autozyg programs and to the Lodscore programs, unless otherwise noted.

`use (locus-by-locus sampling | sequential imputation) for setup`

> There are two setup methods available to find a starting configuration for the meiosis indicators prior to the MCMC: using sequential imputation (with the trait treated as unlinked), or using locus-by-locus sampling (by assuming all markers and trait are unlinked). Sequential imputation is the default method.

`use` *`I`* `sequential imputation realizations for setup`

> When sequential imputation is selected above, this statement specifies the number of sequential imputation samples from which the starting configuration of meiosis indicators is to be selected. The default is 20 iterations.

`set MC iterations` *`I`*

> Required. It specifies the total number of main L- and M-sampling iterations. There is no default number of MCMC iterations; the total number of '`main`' L- and M- sampling iterations must be specified for all Autozyg programs. The total MCMC run length is the sum of the number of burn-in iterations specified by the '`set burn-in iterations`' statement and the number of main iterations specified in '`set MC iterations`'.

`set burn-in iterations` *`I`*

> Burn-in iterations are performed initially, with the trait locus (if any) unlinked to the marker map. The default number of burn-in iterations is specific to each program.

`sample by (scan | step)`

> By default (sample by scan), all loci (L-sampler) or all meioses (M-sampler) are updated successively in an order determined by random permutation. When sampling by step, a single locus (L-sampler) or single meiosis (M-sampler) is randomly selected for updating. `lm_bayes` presently samples by scan only.

`set L-sampler probability` *`X`*

> The L-sampler probability, between 0.0 and 1.0, specifies the probability in each MCMC iteration, of locus-sampling rather than meiosis-sampling. The default is 0.0, that is, to use M-sampler only.

`compute scores every` *`I`* `iterations`

> The default is to score recombinations, total log-probabilities or the Rao-Blackwellized estimator every MCMC iteration. This statement specifies the frequency with which to compute the contributions to the *ibd* scores or the location lod scores.

`check progress` *I* `MC iterations`
: Use this statement to monitor the progress of the program as it is running. It will print out the iteration number every *I*th iteration.

`set global MCMC`
: By default, MCMC is performed component-by-component, and exact computation and/or iid sampling is used on small pedigree components. If this statement is specified, MCMC will be done globally over the data set, and no exact computation will be done. Note that the formerly used `set peeling by component` is eliminated; only global lod scores or analysis will be performed if the global option is chosen. The recommendation is **not** to use this option, but it is retained for compatibility with older examples and data sets.

`set limit for exact computation` *I1*
: This is the limit on the number of meioses in order for exact computation and iid sampling to be used instead of MCMC. The default value is 8; while exact computation for more than 8 meioses is certainly feasible it is often not computationally efficient.

See [Concept Index], page 151, for: MCMC parameter statements, sequential imputation, locus-by-locus sampling, meiosis indicators, MC iterations, burn-in, L-sampler, M-sampler, sample by scan, sample by step.

# 9 Estimating Conditional *ibd* Probabilities by MCMC

See [Concept Index], page 151, for: conditional *ibd* probabilities, identity by descent, *ibd*.

## 9.1 Introduction to `lm_auto`, `gl_auto` and `lm_pval`

See [References], page 149, for details of the cited papers.

The MORGAN programs `lm_auto`, `gl_auto` and `lm_pval` are referred to as "Autozyg" programs, as they estimate autozygosity, or identity by descent (*ibd*). The Autozyg programs use MCMC to infer patterns of *ibd* among members of a pedigree conditional on marker data, and possibly also on trait data. These inferred patterns may then be used in multipoint linkage analyses on large pedigrees where many individuals may be unobserved and exact computation is infeasible. The data are the genotypes at marker loci of observed individuals in pedigrees. For `lm_auto` and `lm_pval` there may also be affectation status (affected / unaffected / unknown) for the trait of interest.

`lm_auto` uses either the old (single-meiosis) or new (multiple meiosis) LM-sampler to realize *ibd* configurations from their conditional distribution given the marker and/or trait data. Given the data, it estimates conditional probabilities of genome sharing patterns (gene *ibd*) among specified haplotypes, often chosen from affected individuals. The marker data are used jointly in the sampling. The resulting *ibd* is either scored marginally at each marker locus, or over windows of a small number of loci.

`gl_auto` also uses either the old or new LM-sampler to realize *ibd* configurations from their conditional distribution given the marker data. By default, a null (no-data) unlinked '`trait`' locus is also included: optionally this may be omitted Section 9.8.8 [Autozyg MCMC parameters and options], page 79. Rather than estimating *ibd* probabilities, `gl_auto` outputs realizations of *ibd* configurations directly to an output file. The output may either be of founder genome labels (FGL or '`gl`') or of the meiosis indicators that determine the FGL, or both. In either case, the output is in a compact format where only changes of FGL or meiosis indicators are recorded, together with the positions of these changes. Positions are in terms of marker indices in the original marker data file, even where markers are subselected for MCMC. These output *ibd* graphs may be used for subsequent analyses of trait data on the pedigrees, without further reference to the pedigree structure or marker data. For further details see [Tho11].

`lm_pval` uses the LM-sampler to provide the conditional distribution of an *ibd* measure, $T$, given marker data. In principle it can be used to provide Monte Carlo estimates of any NPL (Non-Parametric Linkage) statistics for detecting linkage. Trait information provided to the program consists of the list of affected members of the pedigree, provided as the phenotypic status in the pedigree file.

The version of the program `lm_pval` in MORGAN 3.0 (originally released in MORGAN V.2.8) uses the latent p-value distribution of [TG07]. In `lm_pval`, marker data are assumed available on some pedigree members, at some of the marker loci. At each test genome location, the distribution of the *ibd* measure, $T$, conditional on marker data is compared to the unconditional distribution under the null *a priori* distribution uniform over all inheritance vectors. Then quantiles of a latent (fuzzy) p-value distribution are produced. A latent p-value distribution corrected for multiple testing over genome locations is also produced, by scoring the maximum of the *ibd* measure, $T$, over test locations.

Additional programs using latent p-values are under development, including programs for the distribution of latent lod scores obtained in MCMC sampling (`lm_fuzlod`), p-values and randomized tests based on latent lod score statistics (`lm_fzplod`), and randomized confidence sets for the location of a trait locus (`lm_fzconf`). The methods are described by Thompson in [Tho08a]. The program `civil` (see Section 10.1 [Introduction to lm_ibdtests and civil], page 80) also uses latent p-values [DT09].

See [Concept Index], page 151, for: Autozyg programs, `lm_auto` introduction, `gl_auto` introduction, `lm_pval` introduction, Markov chain Monte Carlo, autozygosity, meiosis indicators, identity by descent, descent graph and *ibd* graph, *ibd*, latent p-values, LM-sampler, multiple meiosis sampler.

## 9.2 Sample `lm_auto` parameter file

`lm_auto` uses the parameter file `jv_rep_auto.par` in the `IBD` subdirectory:

```
input pedigree file 'jv_rep.ped'
input seed file '../sampler.seed'
output overwrite seed file '../sampler.seed'

set printlevel 5
select all markers
select trait 1
set trait 1 tloc 1

map gender F  markers            dist 25.5 25.5 25.5 25.5
map gender M  markers            dist 11.2 45.8 11.2 45.8
map gender F  tloc 1  marker 2  dist 12.8
map gender M  tloc 1  marker 2  dist 5.8

set markers 1 2 3 4    allele freqs  .2 .2 .4 .1 .06 .04
set markers 5        allele freqs  .3 .2 .3 .1

set tloc 1  allele freqs  .95 .05

set marker data 5
333   1 3  1 3  1 3  1 3  1 3
331   3 4  3 4  3 4  3 4  3 4
334   2 3  2 3  2 3  2 3  2 3
431   3 4  3 4  3 4  3 4  3 4
531   3 3  3 3  3 3  3 3  3 3

343   1 3  1 3  1 3  1 3  1 3
341   3 5  3 5  3 5  0 0  3 3
344   4 6  4 6  4 6  2 4  2 4
441   3 4  3 4  0 0  3 4  3 4
541   3 3  3 3  3 3  3 3  3 3

set window patterns 0 4
set locus window 3

set component 1 proband gametes  531 1  531 0  331 0  333 1
set component 2 proband gametes  541 1  541 0

set L-sampler probability 0.2
use multiple meiosis sampler
set MC iterations 2000
```

The '`select`' statement is analogous to `genedrop`'s '`simulate`' statement (see Section 5.4.1 [genedrop computing requests], page 33). The statements first specify that all markers will be used. Then the trait ('`1`') for which data will be analyzed is identified and connected to a tloc ('`1`').

The trait values are specified in the pedigree file: the specified pedigree file, `jv_rep.ped`, is a 30-member, two-component pedigree. Since there is no '`input pedigree record trait`' statement in the example parameter file, the default behavior is implemented and so the trait value is listed after the three names and integer gender in the pedigree file. In this example, this is the 5th and final column (2nd integer). Because the trait type is not specified in the parameter file via a '`set trait data`' statement, by default the trait data are '`genotypic`', so that they are are coded as '`1`', '`3`', '`4`' or '`0`', corresponding to trait locus genotypes of '`1 1`', '`1 2`' (or '`2 1`'), '`2 2`' or '`missing`', respectively. In the example, the final individuals of each pedigree component, named 531 and 541, have trait value '`4`'. All other individuals in the file have trait value '`0`'.

The '`map`' statements specify the marker map and trait position in terms of genetic distances (centiMorgan). In this example there are five markers with gender-specific maps. The trait locus position is measured from the marker to its left. In this example, the trait locus for males is between markers 2 and 3 at a distance of 12.8 cM to the left of marker 2 (See See Section 5.4.2 [genedrop mapping model parameters], page 33. The '`set markers`' statements specify the number and frequency of alleles for each marker. In the example, the first four markers each have six alleles (labeled 1–6) with frequencies 0.2, 0.2, 0.4, 0.1, 0.06 and 0.04. The fifth marker has four alleles with frequencies 0.3, 0.2, 0.3 and 0.1. The trait locus has two alleles; alleles '`1`' and '`2`' have frequencies 0.95 and 0.05, respectively.

The '`set marker data`' statement specifies the number of markers to be five. Following the '`set marker data`' statement are genotype data for typed individuals. Alternatively, `lm_auto` can read genotype data from a separate file specified with an '`input marker data file`' statement. Note that in the parameter file the marker-5 allele frequencies do not sum to 1. By default, allele frequencies will **not** be normalized. The implication is that there are other alleles not present in the marker data, whose frequencies therefore need not be listed. If the program encounters in the marker data an allele at this locus other than '`1`' to '`4`', an error will be generated.

The '`set window patterns`' and '`set locus window`' statements instruct `lm_auto` to compute the probabilities that the gametes named in the '`set proband gametes`' statement have a particular *ibd* pattern (also called *state*) jointly across several loci. The '`set locus window`' statement specifies the number of loci to be examined simultaneously, in this case 3. This statement was discussed briefly in the `ibddrop` example: See Section 7.3 [Running ibddrop example and sample output], page 51. The probabilities in the '`IBD`' column of the output specifies whether one of the specified set of patterns holds (‘1’) or not (‘0’) at each of the three loci across the window.

Using the ‘set window patterns’ statement in lm_auto, the user can specify *ibd* patterns of interest over two or more loci. Recall that in the ibddrop windows option, one can can only estimate the probability of two gametes being *ibd* or not. The ‘set window patterns’ statement indicates that we are interested in patterns ‘0’ and/or ‘4’, which correspond to *ibd* patterns ‘1 1 1 1’ and ‘1 1 2 2’, respectively. That is, in component 1, we are interested in the probability that all four of the gametes named in the ‘set proband gametes’ statement are *ibd* across 3-locus windows or that the first and second gametes (maternal and paternal haplotypes of individual 531) are *ibd* and the third and fourth gametes (maternal haplotype of individual 531 and paternal haplotype of individual 333) are *ibd*, but these two pairs are not *ibd* with each other.

Recall the output of the ibddrop program generated when using the parameter file ibd.par. In the section of the program output headed ‘Probabilities of IBD patterns’, each of the *ibd* patterns listed in the leftmost column is associated with a label in the right-most column.

```
Probabilities of IBD patterns

Proband gamete set 1:  541 0  541 1  341 0  343 1

pattern marker-1 marker-2  trait-1 marker-3 marker-4 marker-5    label

1 1 1 1    .0287    .0298    .0310    .0273    .0287    .0298        0
1 1 1 2    .0290    .0275    .0292    .0282    .0302    .0305        1
1 1 2 1    .0132    .0135    .0138    .0140    .0139    .0132        3
```

The ‘set window patterns’ statement in the parameter file for lm_auto expects one or more of these labels, which instruct it to calculate the probabilities of the associated pattern(s). This means that you must determine the labels of the patterns of interest (for example, by running ibddrop), before using lm_auto to estimate multi-locus probabilities.

The ‘set proband gametes’ statement is the key statement for lm_auto. It specifies which haplotypes are to be scored with *ibd* probabilities. The syntax is as follows, where N1, N2, ... are individual ID’s and K1, K2, ... indicate the haplotype as paternal (1) or maternal (0):

```
set [component M proband gametes N1 K1 N2 K2 ...
```

In the example, ‘531 1’ refers to the paternal (1) haplotype of individual ‘531’. The first statement requests scoring both haplotypes of 531, the maternal (0) haplotype of 331, and the paternal (1) haplotype of 333. Note that currently the number of proband gametes to be scored jointly is limited to 10. See Section 7.4 [ibddrop statements], page 54, for more discussion of the ‘set proband gametes’ statement.

As with all of MORGAN’s MCMC-based programs, the user can specify the desired number of MC iterations using the ‘set MC iterations’ statement, the desired number of burn-in iterations using ‘set burn-in iterations’, and the probability that the L-sampler is selected instead of the M-sampler using ‘set L-sampler probability’. In this example, 2000 sampling iterations are to be performed, using the L-sampler 20 percent of the time. These iterations are preceded by burn-in iterations. Because the number of burn-in iterations is not specified, lm_auto will use the default value of 10 percent of the number of main iterations. In practice, one would run the MCMC sampler much longer than 2000 iterations (on the order of 10^5). The '`multiple meiosis`' sampler is requested and the '`set global MCMC`' statement is not used, so MCMC will be performed separately for each pedigree component. For further details of these statements: See Section 8.6 [MCMC parameter statements], page 61.

See [Concept Index], page 151, for: `lm_auto` sample parameter file, trait data, genotypic trait, gender–specific maps, marker data file, window patterns, proband gametes, L-sampler, M-sampler, burn-in.

## 9.3 Running `lm_auto` example and sample output

The syntax for running a MORGAN program is:

```
./<program> <parfile> [> <output file name>]
```

The `lm_auto` example can be run under the subdirectory `IBD`

```
./lm_auto jv_rep_auto.par > auto.out
```

Below are sections of the output file `auto.out`, generated by running `lm_auto` using the parameter file `jv_rep_auto.par`. Note, as for the program `ibddrop`, the exact values of the probability estimates will depend on the value of the random seed. The tables of estimated *ibd* probabilities are given for each component, towards the end of the output. The estimated probabilities of gene *ibd* patterns are given for each marker and for the trait locus (in the map order). In the following extracted output, the MCMC diagnostic information has been omitted.

```
====== IBD scores for component 1 are estimated using MCMC ======

      Proband gamete set 1:  531 1  531 0  331 0  333 1

      pattern marker-1 marker-2 trt-geno marker-3 marker-4 marker-5    label

      1 1 1 1    .2345    .3375    .3655    .2945    .2450    .1870        0
      1 1 1 2    .1165    .1815    .2250    .1785    .1425    .0940        1
      1 1 2 1    .1540    .2025    .2595    .1605    .1300    .1070        3
      1 1 2 2    .0135    .0215    .0270    .0195    .0095    .0080        4
      1 1 2 3    .0260    .0270    .0435    .0250    .0210    .0195        5
      1 2 1 1    .0190    .0110    .0010    .0115    .0200    .0275        6
      1 2 1 2    .0715    .0430    .0130    .0600    .0665    .0720        7
      1 2 1 3    .0560    .0250    .0090    .0275    .0420    .0520        8
      1 2 2 1    .0345    .0205    .0030    .0265    .0380    .0310        9
      1 2 2 2    .0925    .0435    .0120    .0685    .1060    .1410       10
      1 2 2 3    .0425    .0285    .0105    .0325    .0410    .0515       11
      1 2 3 1    .0245    .0125    .0040    .0115    .0210    .0275       12
      1 2 3 2    .0620    .0300    .0130    .0515    .0670    .0885       13
      1 2 3 3    .0115    .0020    .0040    .0125    .0125    .0185       14
      1 2 3 4    .0415    .0140    .0100    .0200    .0380    .0750       15

   Probabilities of IBD for pattern set for windows of 3 loci
```

```
          Proband gamete set 1

          Pattern set:   0  4

            IBD   wndw 1 wndw 2 wndw 3 wndw 4

          0 0 0   .4955  .4635  .4630  .5410
          0 0 1   .0810  .0815  .0520  .0740
          0 1 0   .0345  .0415  .0375  .0430
          0 1 1   .1410  .0545  .0550  .0280
          1 0 0   .0495  .0515  .1520  .1125
          1 0 1   .0150  .0110  .0190  .0180
          1 1 0   .0280  .1295  .0930  .1085
          1 1 1   .1555  .1670  .1285  .0750


    ====== IBD scores for component 2 are estimated using MCMC ======

       Probabilities of IBD patterns

          Proband gamete set 1:  541 1  541 0

          pattern marker-1 marker-2 trt-geno marker-3 marker-4 marker-5    label

             1 1    .7000    .8760    .9580    .8165    .6570    .4670        0
             1 2    .3000    .1240    .0420    .1835    .3430    .5330        1
```

Interpretation of these results is similar to that of `ibddrop` See Section 7.3 [Running ibddrop example and sample output], page 51. Briefly, the probabilities are summarized by *ibd pattern*. A pattern is a series of integers, one representing each gamete listed in the '`set proband gametes`' statement. The order of gametes in the output file patterns is the same as the order in which the gametes were listed in '`set proband gametes`'. Numbers that are the same indicate gametes that are *ibd*. For instance, in the first row of the table above, the pattern is '`1 1 1 1`', which means that the values in the first row represent probabilities that all four gametes are *ibd* at each marker locus and at the trait locus. Likewise, '`1 2 1 1`' means gametes 1, 3, and 4 are *ibd* while gamete 2 is not *ibd* with the others; '`1 2 3 4`' means all four gametes are not *ibd*.

The second table in the above output is a result of the window size and *ibd* pattern statements in the parameter file. Its interpretation is similar to the output of `ibddrop` when statement '`set locus window`' was used, See Section 7.3 [Running ibddrop example and sample output], page 51. Recall that in `ibddrop`, the values in the '`IBD`' column of the output indicate whether the two gametes specified in the '`set proband gametes`' statement are *ibd* (indicated by a '`1`') or not (indicated by a '`0`'). With `lm_auto`, the user can specify additional *ibd* patterns of interest over two or more gametes. In this example, the parameter file `jv_rep_auto.par` includes the statement '`set window patterns 0 4`', which indicates that we are interested in *ibd* patterns '`0`' and '`4`', corresponding to '`1 1 1 1`' and '`1 1 2 2`', respectively, as discussed in the previous section. That is, we would like to know the probability that either all four gametes are *ibd* or that the first two are *ibd* and the second two are *ibd*, but the pairs are not *ibd* with one another for each window of three loci. Consequently, interpretation of the ‘IBD’ column of the lm_auto output is as follows. The row headed by ‘0 0 0’ gives probabilities that the gametes do not follow either of the two *ibd* patterns of interest at all three loci for each window. The row headed by ‘0 0 1’ gives probabilities that the gametes do not follow either of the two *ibd* patterns of interest at the first two loci in the window, but at the third loci either all four gametes are *ibd or* the first two are *ibd* and the last two are *ibd*, but the pairs are not *ibd* with one another.

In this section of the lm_auto output, the order of the marker and trait loci is the same as in the table of results for each locus; that is, the map order. In this example, the trait locus was between markers 2 and 3. Therefore, the windows are as below:

| **window** | **loci** |
|---|---|
| wndw 1 | marker 1, marker 2, trait |
| wndw 2 | marker 2, trait, marker 3 |
| wndw 3 | trait, marker 3, marker 4 |
| wndw 4 | marker 3, marker 4, marker 5 |

For more information regarding the MCMC parameters and diagnostic output, See Section 8.5 [MCMC computational options], page 59.

See [Concept Index], page 151, for: running lm_auto example, lm_auto sample output, ibd pattern, proband gametes.

## 9.4 Sample gl_auto parameter file

The parameter file of the gl_auto example is in the ‘IBD’ subdirectory of MORGAN_Examples. Like the earlier markerdrop example (see Section 6.2 [Sample markerdrop parameter file – conditional on trait], page 37), the gl_auto uses the 3-component pedigree with a total of 73 individuals, ped73.ped, and the corresponding 10-marker data, ped73.marker.missing. Both these files are in the main MORGAN_Examples directory; that is ‘..’ relative to the parameter file.

The gl_auto parameter file is given here in full, as it contains several options not so far encountered in this tutorial, as well as some statements specific to this program.

```
input pedigree file '../ped73.ped'
input seed file '../sampler.seed'
output overwrite seed file '../sampler.seed'
input marker data file '../ped73.marker.missing'
select all markers

set printlevel 5                # Include everything in the output file.

select trait 1
input pedigree record trait 1 integer 7     # dummy (0) trait values
set trait 1  tloc 11                        # Connect trait and tloc
map tloc 11 unlinked                        # Trait locus is unlinked.
```

```
set tloc 11 allele freqs 0.5 0.5

# Monte Carlo setup and requests.
# Use the default option of sampling by component.
# Specify which meiosis sampler is to be used.

use multiple meiosis sampler
set limit for exact computation 12
set MCMC markers only                       # Do MCMC for markers only
use sequential imputation for setup
use 5 sequ impu realiz for setup
sample by scan                              # Default: for clarity only
set L-sampler probability 0.5

## For real analyses, recommended number of iterations is of order 10^5
set MC iterations 2000                 # For golds and checks only.
set burn-in iterations 15              # For golds and checks only.

# Specify what type of output is desired (or both).
# Specify the desired scoring interval.

output founder genome labels
output overwrite scores file './ped73_glauto.fgl`
output meiosis indicators
output overwrite extra file './ped73_glauto.meio'
output scores every 30 scored MC iterations
```

The first block of statements specify data and seed files in a way that should by now be familiar. Note all markers are selected; all will be used in the MCMC and output *ibd* graphs. The second block defines a trait and tloc combination. However the trait expected by `gl_auto` is a dummy trait consisting entirely of 0 values for '`unobserved`'.

The Monte Carlo requests are those used by most of the MCMC-based programs: See Section 8.6 [MCMC parameter statements], page 61. Unlike earlier examples, the multiple meiosis sampler is used, and sampling is (by default) by pedigree component as'`global MCMC`' is not requests. A limit of 12 meioses is set for exact computation; in fact even the smallest 11-member pedigree component has 14 meioses, so MCMC will be done on each of the three components. Other MCMC requests are standard and similar to those used in the `lm_auto` example. Note again than many more MCMC scans (and associated burn-in) would be done in a real example.

The final five parameter statements are specific to the `gl_auto` program. This program outputs *ibd* graphs or meiosis indicators (or both) to an output file. The *ibd* graph output is sent to the output scores file, so if this output is requested the name must be given. Meiosis indicators are sent to the output extra file, so if meiosis indicators are requested this file name must be given. It is strongly recommended that the `overwrite` option be used for both these outputs to avoid appending to previous runs of the program. Finally the the output scoring frequency is given. Here every MCMC iteration is '`scored`' (the default), but *ibd* graphs and meiosis indicators are computed and output only every 30 iterations. Note that in pedigrees with multiple components, the same number of *ibd* graphs are generated on each component. On small components where exact IID realizations are generated, every realization is output; the number of such realizations is the same as the number output on any component for which MCMC is used.

See [Concept Index], page 151, for: `gl_auto` sample parameter file, multiple meiosis sampler, founder genome labels, exact computation, *ibd* graph.

## 9.5 Running `gl_auto` example and sample output

The syntax for running a MORGAN program is:

```
./<program> <parfile> [> <output file name>]
```

The `gl_auto` example can be run under the subdirectory `IBD`

```
./gl_auto ped73_glauto.par > ped73_glauto.out
```

The output file `ped73_glauto.out`, generated by running `lm_auto` using the parameter file `ped73_glauto.par` is actually of little interest, but should be checked to see that the program has interpreted the data as expected. The output consists of summaries of the input pedigrees, and several MCMC diagnostics. As for the `ibddrop lm_auto` programs, the exact values of the probability estimates will depend on the value of the random seed; note that this parameter will output final seeds overwriting the previous input seed file.

The important output is contained in the '`output scores file`', which the parameter file specified to be `ped73_glauto.fgl`. Note that if you run the example with different seeds your results will differ from the example output given below. Note also that the parameter file specifies that any previous output scores file of the same name will be overwritten; if you want to save an earlier one, rename it! The output scores file contains 9636 lines, which are 66 (2000/30) realizations of *ibd* graphs. There are three components, with 47, 11 and 15 individuals, and each individual is output on two lines – a maternal chromosome and a paternal chromosomes. The first 94 (47+47) lines is the first output realization on the first component. The other 65 realizations follow for a total of 94 times 66 (6204) lines. There are then 66 realizations of the second component ((11+11) times 66 = 1452 lines), and finally the realizations on the third component ((15+15) times 66 = 1980 lines). The total line-count of the file is 9636.

For the seeds on this run, the first few lines of the file are shown as

```
101 0 1 0
101 0 2 0
102 0 3 0
102 0 4 0
201 0 4 3 5 3  8 4  9 3
201 0 1 1 2 2
```

The first column is the name of the individual (2 lines for each individual). The zero second column may be ignored. The third column is the initial (first marker) FGL, and for 101 and 102 there are no FGL-changes as they are founders. Individual 201 is the offspring of 101 and 102. His maternal chromosome consists of segments of FGL 3 and 4: it is initially 4 and there are 3 switches. At marker 5 the switch is to FGL 3, at marker 8 back to 4, and at 9 back to 3 again. Individual 201's paternal chromosome has only 1 switch, switching from 1 to 2 at marker 2.

Lower down the pedigree there may be more FGL and more switches. For example for the maternal chromosome of 407 we have

```
407 0 4 4 5 3  8 4  9 3  10 6
```

The initial FGL is 4, and there are 4 switches: to 3, 4, 3, and 6 at markers 5, 8, 9, and 10. respectively.

Next consider the meiosis output in the '`output extra file`' which was specified to be `ped73_glauto.meio`. This file contains a line for every meiosis sampled. Typically this is 2 lines for every non-founder individual, but a single child of a founder provides only one meiosis. In the three components of size 47, 11, and 15, there are 13, 4 and 5 founders respectively, and hence 34, 7, and 10 non-founders, but only 65, 14, and 20 meioses sampled. Hence the 66 output realizations give 4290, 924, 1320 lines for a total of 6534 lines.

The first few lines of the output file are

```
201 2 0 0 1 0
201 2 1 0 0 1 5
202 3 0 0 0 0
202 3 1 0 1 3 4 7 9
301 5 0 0 1 0
301 5 1 0 0 0
302 6 0 0 1 1 8
302 6 1 0 0 1 8
```

The first two columns are the name and the internal MORGAN 0-origin index of the individual, and the third column specifies whether this is the maternal (0) or paternal (1) meiosis of the individual. The fourth column refers to the unlinked null locus, and is here null (0) as this locus was not sampled by the MCMC. The fifth column gives the meiosis indicator at the first marker, and the sixth the number of changes across the chromosome. The final integers, in number corresponding to the number of switches, give the change points. Unlike the *ibd* graph file, no additional details of the changes are needed. since changes are from 0 to 1, or 1 to 0,

Some important points are that, first, there are programs to process and use this *ibd* graph and meiosis output; the user should not be concerned with the details. However, it is important that the user knows how many graphs they have generated, and whether they are done by component or globally. Until processing programs with multi-component capability have been released, it is recommended to separate the graphs by component. In general. if sampling is by component, then output is for each component, for each realization, for each individual/meiosis. If sampling is global, then output is for each realization, for each component, for each individual/meiosis. Sampling by component is normally recommended: "global MCMC" is maintained mainly for backwards compatibility.

Finally, it should be seen that the output format is compact. Here we have only 10 markers, so could have output by marker. However, the files would be no larger on the same pedigrees if we have many thousands of markers. The number of switches depends on the length of chromosome and pedigree depth, not on the marker count. (See [Tho11]).

See [Concept Index], page 151, for: running `gl_auto` example, `gl_auto` sample output.

## 9.6 Sample `lm_pval` parameter file

Files for `lm_pval` may be found in the `TraitTests` subdirectory of `MORGAN_Examples`. The parameter file, `ped73_pval.par` is similar to the parameter file used for `lm_auto`. An abbreviated version of `ped73_pval.par` is given below:

```
input pedigree file '../ped73.ped'

input pedigree record trait 1 integer 3
select trait 1

input seed file '../sampler.seed'

input marker data file '../ped73.marker.missing'
select all markers

set L-sampler probability 0.2
set MC iterations 2000
```

For `lm_pval`, markers are selected, but no trait locus is selected. Therefore, no '`map tloc marker`' statements are included, and no '`set traits tlocs`' statement is included. The file `ped73.marker.missing` contains the marker map and genotypes, and is accessed by the statement '`input marker data file`'.

Pedigree members affected with the disease must be specified when using `lm_pval`. The set of affected individuals is determined implicitly by using trait data. The statement '`select trait 1`' instructs the program to determine the affected individuals by using the trait data for trait 1 in the pedigree file. The statement '`input pedigree record traits`' is needed to define the correspondence between trait numbers and integers in the pedigree record, so that the program knows where to find the desired trait data. The trait data in this example are (by default) genotypic: the `lm_pvals` program treats both homozygotes and heterozygotes for the disease allele '2' as affected.

See [Concept Index], page 151, for: `lm_pval` sample parameter file.

## 9.7 Running `lm_pval` example and sample output

Under the subdirectory `TraitTests`, run the `lm_pval` example by typing:

```
./lm_pval ped73_pval.par > pval.out
```

A portion of the output giving latent (fuzzy) p-values is below. See `pval.out` for the entire output file.

```
 Combined distribution of fuzzy p-values, by locus:
pval maxim  marker-1  marker-2  marker-3  marker-4  marker-5  marker-6  marker-7
            marker-8  marker-9 marker-10
0.00 0.000     0.000     0.000     0.000     0.000     0.000     0.000     0.000
               0.000     0.000     0.000
0.01 0.002     0.015     0.008     0.000     0.001     0.002     0.000     0.002
               0.000     0.000     0.000
0.02 0.006     0.025     0.019     0.000     0.002     0.002     0.000     0.007
               0.000     0.000     0.000
```

```
   0.03 0.008     0.035     0.030     0.003     0.006     0.003     0.000     0.017
                  0.000     0.000     0.000
   0.04 0.012     0.045     0.041     0.007     0.009     0.004     0.000     0.028
                  0.000     0.000     0.000
   0.05 0.015     0.055     0.051     0.011     0.013     0.006     0.002     0.039
                  0.000     0.000     0.000
```

The output table shows the cumulative distribution of the latent (fuzzy) p-values generated at each marker position, as well as the cumulative distribution of the maximum latent p-value over the markers. These distributions are over the latent inheritance patterns sampled, given the marker data. That is, for each value of '`pval`' in the left column, the table gives the proportion of sampled inheritance vectors at each marker that yield a p-value less than '`pval`'. In the last row of the example output, when pval = 0.05, 0.6% of the realizations have a p-value less than 0.05 at marker-5; at marker-7 this value is 3.9%. Overall, 1.5% of the realizations have a maximum p-value over the markers that is less than pval = 0.05 (shown in the second column labeled 'maxim').

Recall again, that exact values in the output will depend on the random seed. In the case of a relatively short run of `lm_pval` there may be substantial differences in the estimated latent p-value distributions.

For more information regarding the MCMC parameters and diagnostic output: See Section 8.5 [MCMC computational options], page 59.

See [Concept Index], page 151, for: running `lm_pval` example, `lm_pval` sample output, fuzzy p-value.

## 9.8 Autozyg statements

Many of the `lm_auto` and other statements following are also used for the location lod scores programs. See Section 11.9 [Location lod scores statements], page 111.

See [Concept Index], page 151, for: Autozyg statements, `lm_auto` statements, `gl_auto` statements, `lm_pval` statements.

### 9.8.1 Autozyg computing requests

`select [chromosome` *I*`] all markers`
: This statement selects all markers on the chromosome for the computation; if not all markers are to be used, use the next statement.

`select [chromosome` *I*`] markers` *J1 J2* `...`
: This alternate form of the '`select markers`' statement specifies a subset of the markers.

`select [chromosome` *I*`] all markers except` *J1 J2* `...`
: This new form of statement allows the user to specify markers on the chromosome to be omitted in computation.

`select trait` *K*
: This statement names the selected trait, and is used in both `lm_auto` and `lm_pval`. For the former, this names the trait to be used in subsequent mappings to trait loci. For the latter, it is used to determine disease affection status of individuals.

`set traits` *K1* `...` `tlocs` *L1* `...`
: This statement establishes the correspondence between traits and trait loci in `lm_auto`. Note that this statement is not applicable for `lm_pval`.

`map tlocs` *L1* `... unlinked`
: Use this statement for `lm_auto` if the trait locus is unlinked. *L1* is the trait locus number.

`check marker consistency [only]`
: Before running a MORGAN program which uses marker data, the setup routines check that the marker data for the selected markers are consistent with Mendel's first law and stated pedigree information. In the absence of this parameter statement, the program will terminate with the first error found. If this statement is included, the program continues checking the rest of the markers for further inconsistencies and provides details regarding each inconsistency detected. The program terminates only after checking all the selected markers. In the absence of this statement, if no marker data inconsistencies are found, the program continues with any requested analyses. If the '`check marker consistency only`' statement is used, the program will terminate after checking all the markers; this may be useful in any initial phase of checking the data.

These alternative terminations for '`check marker consistency`' are given in tabular form:

| | No error is present | Some error(s) present |
|---|---|---|
| No statement | No termination | Terminates at first error |
| check markers consist | No termination | After checking all markers |
| check mark consist only | After checking all markers | After checking all markers |

See [Concept Index], page 151, for: Autozyg computing requests consistency (Mendelian) of marker data.

## 9.8.2 Autozyg file identification statements

All of the general MORGAN file identification statements can be used with the `Autozyg` programs. For a list of these statements, see Section 2.3 [File identification statements], page 11. Some additional file identification can be used by `Autozyg` programs:

`output score file filename`
: This file can be used by `lm_auto` to save interim cumulative scores of *ibd* probabilities. The `gl_auto` program expects expects an '`output score file`' to be specified, since it uses it for its primary output that is then input to other programs.

`input rescue file filename`
: A rescue file may be used to continue an `lm_auto` run instead of restarting at the beginning. This file contains intermediate data, which is periodically saved when an output rescue file has been specified in a preceding run. (Note this option is not currently used/tested/)

`output rescue file filename`
: This statement, which is optional for `lm_auto`, specifies the periodic dumping of intermediate results to files that may be used to restart the program midstream.

Data are written alternately to files with '`1`' and '`2`' appended to the file name. (Note this option is not currently used/tested/)

See [Concept Index], page 151, for: Autozyg file identification statements, score file, rescue file.

### 9.8.3 Autozyg pedigree file description

Both Autozyg programs use the general MORGAN pedigree file description statements; see Section 2.9 [Pedigree file description statements], page 15.

See [Concept Index], page 151, for: Autozyg pedigree file description.

### 9.8.4 Autozyg output file description

Three output file description statements are expected by `gl_auto` and one additional one is optional for `lm_auto`.

`output (founder genome labels | meiosis indicators)`
: This statement is expected by '`gl_auto`'. It tells the program whether its output is to be in the form of founder genome labels or meiosis indicators (or both). The `founder genome labels` are output to the `output scores file` and the `meiosis indicators` are output to the `output extra file` so these filenames must be specified if the output is requested.

`output rescue data` *I* `iterations`
: This statement can be used to specify the frequency of dumping program data if an output rescue file is specified. (Note this option is not currently used/tested/)

`output scores every` *I* `scored MC iterations`
: Note that this output option is in terms of scored iterations; not every MCMC iteration may be scored; see the *compute scores* statement. This statement is expected by '`gl_auto`', since this program uses the d output score file for its primary output of founder genome labels or meiosis indicators (see the '`output`' statement above. If an output score file is specified, but this statement is not present, '`lm_auto`' writes the score file at the conclusion of the last iteration only (see '`set MC iterations`' statement).

See [Concept Index], page 151, for: Autozyg output file description, founder genome labels.

### 9.8.5 Autozyg mapping model parameters

- For specifying the marker map, see Section 5.4.2 [genedrop mapping model parameters], page 33.
- To specify a trait map for `lm_auto`, see Section 5.4.2 [genedrop mapping model parameters], page 33.

  The trait number specifies its position in the pedigree record; you may need to use the '`input pedigree record traits`' statement (see Section 2.9 [Pedigree file description statements], page 15) to establish the correspondence between trait numbers and integers in the pedigree record.

See [Concept Index], page 151, for: Autozyg mapping model parameters.

### 9.8.6 Autozyg population model parameters

- See Section 5.4.3 [genedrop population model parameters], page 34, for statements specifying the allele frequencies for the markers and traits.

`set [chromosome I] marker names N1 N2...`

This statement, which is optional for both `lm_auto`, `gl_auto` and `lm_pval`, specifies the names of the markers in the order of their position in the marker data file, for example, '`set marker names D1S306 D1S249 D1S245 ....`'.

See [Concept Index], page 151, for: Autozyg population model parameters.

### 9.8.7 Autozyg computational parameters

- See Section 7.4 [ibddrop statements], page 54, for statements specifying the proband gametes and locus window for `lm_auto`.
- See Section 7.4 [ibddrop statements], page 54, for the statement for setting the seeds for the LM-sampler.

`set [chromosome I] markers K data N1 M11 M12 ... [N2 M21 M22 ...] ...`

Individuals with at least one observed marker are named, together with their marker genotypes. The number of allele pairs for an individual is the same as the number of markers mapped on the chromosome. Marker loci not observed for an individual are given alleles '`0 0`'. (Those individuals with no observed markers may but need not be included in this statement.)

In the example, there are 5 markers mapped for the chromosome:

```
set markers 5 data   343   1 3  1 3  1 3  1 3  1 3
                     331   3 4  3 4  3 4  3 4  3 4
                     334   2 3  2 3  2 3  2 3  2 3
                     431   3 4  3 4  3 4  3 4  3 4
                     531   3 4  3 3  0 0  3 3  3 3
```

In this example five individuals have some observed marker data, but individual '`531`' is unobserved at marker 3.

`set [component M] [scoreset N] proband gametes N1 K1 N2 K2 ...`

This statement is required for `lm_auto`. One or more scoring sets may be given for each pedigree component, where a scoring set consists of two or more haplotypes. If there is more than one set for the component, each set is assigned a number 1 or greater. The maximum number of haplotypes in each set is limited to 10, due to computer memory considerations.

Pairs of names and meiosis indicators are given, with 0 indicating maternal inheritance and 1 indicating paternal inheritance. In the example, there are two sets for the component:

```
set component 1 scoreset 2 proband gametes 531 1 531 0 331 0 331 1
set component 1 scoreset 4 proband gametes 561 1 362 0 364 1
```

At least one proband gamete set must be specified when running `lm_auto`.

`set [chromosome I] locus window K`

This statement is optional for `lm_auto` and gives the window size (number of loci) for which the multi-locus *ibd* probabilities are scored. If no size is given, each locus is scored separately.

`set [component` *`M`*`] [scoreset` *`N`*`] window patterns` *`L1`*`...`

This statement is a companion to '`set locus window`' and is required for `lm_auto` when the window option is chosen. It identifies the *ibd* patterns to be jointly scored for the proband gamete set *N* assigned by the '`set proband gametes`' statement. A prior run, with the same proband gametes, but without the window option is needed to select the *ibd* patterns. That is, the user is required to list *ibd* patterns of interest by label; the labeling of the patterns is not obvious without first running `lm_auto`. In the example, we were interested in *ibd* patterns '`1 1 1 1`' and '`1 1 2 2`', which are assigned labels '`0`' and '`4`', respectively, in the output table headed '`Probabilities of IBD patterns`'. One needs to run `lm_auto` to obtain these labels.

`set trait` *`K`* `data (genotypic | discrete | quantitative)`

Trait data are specified as genotypic, discrete (phenotypic), or quantitative (continuous). They may also be specified as '`discrete with covariate`' and as '`discrete with liability`' With a genotypic trait, the trait locus genotype can be inferred from the trait value. There are four possible trait values: '`0`' = missing, '`1`' = homozygous for allele 1, '`3`' = heterozygous, and '`4`' = homozygous for allele 2. There are three possible trait values with a discrete (or phenotypic trait): '`0`' = missing, '`1`' = unaffected, and '`2`' = affected. If a discrete trait is chosen, the next statement, '`set incomplete penetrances`', must be included. With a quantitative trait, a missing value is denoted as a real number with integer portion '`999`'. For example, '`999`', '`999.3`' and '`999.543`' all mean '`missing`'. The default trait type is genotypic.

`set traits` *`K1`* `... for tlocs` *`L1`* `... incomplete penetrances` *`X1 X2 X3`*

This statement is required for discrete trait data. Penetrances (probability of expressing the trait) are provided for the (1 1), the (1 2), and the (2 2) genotypes, respectively.

`input pedigree record trait` *`K`* `integer pairs` *`J1 J2`*

For liability classes or age-or-onset the trait data consists of the qualitative trait status (*J1*) and an additional covariate integer (*J2*) which may specify a liability class or age of onset.

`set trait data` *`K`* `discrete with liability`

This statement allows trait data to be input as a pair of integers, the trait status and the liability class.

`set` *`K`* `liability classes with penetrances` *`X11 X12 X13`* `...` *`XK1 XK2 XK3`*

This statement provides the penetrances for each of the three `tloc` penetrances for each of the *K* liability classes. There is a hard maximum of 24 liability classes.

`set trait` *`K1`* `data discrete with covariate`

The statement allows a covariate such as age-of-onset to be input as a part of the trait data. The program then expected that the `input pedigree record trait` statement will specify `integer pairs`.

`set standard deviations for genotypes` *`X1 X2 X3`*

This statement specifies the standard deviation of age-of-onset for each of the three `tloc` genotypes. The mean age-of-onset is specified via the `set traits for tlocs genotype means` parameter statement (see Section 5.4.3 [genedrop population model parameters], page 34).

`set width of ages of onset window` *`X1`*

This number is used in the age-of-onset penetrance function, allowing actual onset to be up to *X1* units (usually years) earlier than the recorded age of onset.

`select trait` *`K`*

`lm_pval` needs to know which members of the pedigree are affected with the disease. Discrete or genotypic data for the selected trait is used to determine the disease affection status of the individuals. Here, `lm_pval` is to determine the affected individuals from the trait data in the pedigree file. A trait genotypic code of 3 (genotype (1 2) or (2 1)) or 4 (genotype (2 2)) indicates an affected individual. The trait number, *K*, determines the position of this genotypic code in the pedigree records (see Section 2.9 [Pedigree file description statements], page 15).

See [Concept Index], page 151, for: Autozyg computational parameters, marker data, missing marker data, scoreset, proband gametes, locus window, scoreset, window patterns, trait data, genotypic trait, discrete trait, quantitative trait, incomplete penetrance, affected individuals.

### 9.8.8 Autozyg MCMC parameters and options

- All the statements described in see Section 8.6 [MCMC parameter statements], page 61, for specifying the MCMC parameters are used by the Autozyg programs.

Please see that section for details regarding:

```
use (locus-by-locus sampling | sequential imputation) for setup
use I sequential imputation realizations for setup

set MC iterations I
set burn-in iterations I
sample by (scan | step)
set L-sampler probability X

check progress I MC iterations
```

One additional statement is specific to the `gl_auto` program:

`set MCMC markers only`

This statement will cause the `gl_auto` program to do MCMC only for the markers, and not for the assumed unlinked no-data '`trait`' locus. This allows for identical MCMC with the `lm_linkage` lod score program if the same MCMC options are used, and hence for comparison of results among these programs.

See [Concept Index], page 151, for: Autozyg MCMC parameters and options.

# 10 Estimating *ibd* Based Test Statistics by MCMC

See [Concept Index], page 151, for: *ibd*-based tests.

## 10.1 Introduction to `lm_ibdtests` and `civil`

See [References], page 149, for details of the cited papers.

The program `lm_ibdtests` uses identity-by-descent (*ibd*) based and likelihood-ratio based statistics to construct linkage detection tests. The current version allows only discrete trait data (affected or unaffected or unknown phenotypic status).

The *ibd* scoring approach involves construction of an *ibd* measure (T) that is a function of the inheritance vectors and affectation status of the individuals in pedigrees. The program uses realizations of the inheritance vectors conditional only on the marker data (Y) to compute a Monte Carlo estimate of the test statistic E(T|Y). Four different *ibd* measures are implemented in the program. Two of these measures, T=Slambda and T=Saffunaff (developed by Saonli Basu), allow incorporation both of affected and of unaffected individuals in the analysis. The test statistic is used to test the null hypothesis of no linkage between the trait and a set of markers. For this approach, two different testing options have been implemented; one is a normality-based test and the other is a permutation test. The permutation test keeps the observed marker data unchanged and permutes the affectation status. In the normality-based test, test statistics (T=Spairs, for example) are computed for each realization and averaged over realizations. The program then reports the p-values from each test at the marker loci. For more details of these methods, see [Bas08].

A new (lambda,p) model has been implemented in `lm_ibdtests`. The (lambda,p) model models the trait-dependent segregation of inheritance vectors at a locus given the trait data on individuals and constructs a chi-square test for linkage detection. The (lambda,p) model incorporates both affected and unaffected individuals in the analysis. The delta model is also implemented in the program. The current version of `lm_ibdtests` only allows the *ibd* measure T=Spairs in the delta model set-up. The program returns the p-values of the likelihood-ratio statistics under each of these two models. For a detailed description of the (lambda,p) and delta models, see [Bas10]. For a real data analysis using `lm_ibdtests`, see [Sie05].

The program `civil` is due to Yanming Di, see [DT09]. It is still in beta-test version. The program performs marginal and conditional inheritance vector tests for linkage detection and localization. The name `civil` is an acronym for Conditional Inheritance Vector test In Linkage analysis.

In an inheritance vector test, the test statistic is a score that measures the connection between the observed trait values and the inheritance vector at the test position. Excess such connection provides evidence for genetic linkage. `civil` implemented two such scores: a variance component type score (the vc-score) and a score developed by Yanming Di (the w-score).

`civil` computes marginal and conditional test p-values using Monte Carlo method: to approximate the null test statistic distributions, the program will hold trait values fixed and resample the inheritance vectors. The inheritance vectors along a chromosome should follow a Markov Chain distribution in genomic regions absent of causal genetic variants. In a marginal test, the null inheritance vectors are sampled from the marginal distribution of the Markov Chain, which is uniform over the set of all possible inheritance vectors (see Section 9.1 [Introduction to lm_auto gl_auto and lm_pval], page 63). In a conditional inheritance vector test, the inheritance vectors are sampled from the conditional distribution the inheritance vector at the test position given the observed inheritance vectors at the two conditioning positions, as determined by the Markov Chain distribution.

A significant conditional test result provides linkage localization information: it suggests that linkage signal exists in the region bounded by the two conditioning positions, and the conditional p-value gives the false positive probability. A significant marginal test result does not allow such interpretation. For conditional tests, there is a trade-off between power and precision. When the two conditioning positions are more far apart, the conditional test will be more powerful, but a significant conditional test result will provide less precise localization information.

See [Concept Index], page 151, for: `lm_ibdtests` introduction, `civil` introduction, vc-score and w-score.

## 10.2 Sample `lm_ibdtests` parameter file

The example parameter file for `lm_ibdtests`, `ped73_ibdt_IBD.par`, may be found in the `TraitTests` subdirectory of `MORGAN_Examples`. Several lines in the example parameter file have been explained in previous sections of the tutorial, only the sections requiring additional explanation are shown below.

```
sample by scan
set L-sampler probability 0.5
set burn-in iterations 1000
check progress MC iterations 1000

compute ibd statistics
set ibd measures Spairs Srobdom
set ibd tests norm permu
set ibd permutations 999

compute scores every 100 iterations
```

The statement '`sample by scan`' indicates that all loci or all meioses are updated successively in an order determined by random permutation. The alternative '`sample by step`' updates only one locus (L-sampler) or one meiosis (M-sampler) in each iteration. The '`set L-sampler probability`' statement specifies that an L-sampler step/scan will be used at each MCMC iteration with probability 0.5: otherwise the single-meiosis M-sampler will be used. The '`set burn-in iterations`' statement specifies 1000 iterations to be performed initially, with one trait locus (if any) unlinked to the marker map. The '`check progress`' statement instructs the program to print the current iteration number to `stdout` every 1000 iterations.

The '`compute ibd statistics`' statement must be included in the parameter file when running `lm_ibdtests`. The next line instructs the program to use Spairs and Srobdom to perform the *ibd* tests. The '`set ibd tests`' command calls for both normal and permutation tests to be run. The next line is needed since permutation test were requested in the previous line; it specifies how many permutations are to be used in the calculations. In this case, the default (999) is specified; it is recommended that at least 50 permutations are used. The last line in the parameter file is used to specify when to compute scores, the default is every MCMC iteration.

See [Concept Index], page 151, for: sample parameter file for `lm_ibdtests`.

## 10.3 Sample `lm_ibdtests` output

Under the subdirectory `TraitTests`, run the example with the following command

```
./lm_ibdtests ped73_ibdt_IBD.par
```

The part of the output that tabulates test statistics and p values is shown below. The upper table provides the permutation-test p-values for each of the two test statistics Spairs and Srobdom at each of the 10 marker-locus positions, these positions being given for both the male and female genetic maps. It is apparent that there is no significant association of the trait with any of these marker positions; the p-values at markers 5 and 6 are somewhat smaller, but do not achieve (e.g.) a 0.05 significance level. The lower table gives the same result, but this time using a Normal distribution approximation to obtain the p-value. In this case the standardized (N(0,1)) value of the test statistic is given, as well as the corresponding p-value. Again there are no significant results in this small example. There is a broad qualitative correspondence between the p-values of the two tables, but the results are not close. This may be due to the small number of permutations used, or, more likely, due to the inadequacies of the Normal approximation.

```
************************************
p Value for Permutation Test for IBD
************************************

           pos(Haldane cM)   Spairs  Srobdom
    locus     male  female  p-value  p-value

 marker-1    0.000   0.000   0.9020   0.9300
 marker-2   10.000  10.000   0.8780   0.8450
 marker-3   20.000  20.000   0.8130   0.7800
 marker-4   30.000  30.000   0.5080   0.5190
 marker-5   40.000  40.000   0.2550   0.2480
 marker-6   50.000  50.000   0.2950   0.2510
 marker-7   60.000  60.000   0.3850   0.5090
 marker-8   70.000  70.000   0.5100   0.6660
 marker-9   80.000  80.000   0.6610   0.7750
marker-10   90.000  90.000   0.5640   0.7470

*******************************
p Value for Normal Test for IBD
*******************************

           pos(Haldane cM)
   locus     male  female   Spairs p-value  Srobdom p-value
```

```
    marker-1    0.000   0.000  -0.7843  0.7951  -0.2867  0.6167
    marker-2   10.000  10.000  -0.9574  0.8166  -0.3841  0.6567
    marker-3   20.000  20.000  -1.1825  0.8816  -0.2260  0.5692
    marker-4   30.000  30.000  -0.6437  0.7381  -0.1272  0.5552
    marker-5   40.000  40.000   0.2478  0.4103   0.0986  0.4743
    marker-6   50.000  50.000  -0.2270  0.5752  -0.3275  0.6252
    marker-7   60.000  60.000  -0.1503  0.5612  -0.3514  0.6437
    marker-8   70.000  70.000  -0.3096  0.6372  -0.3587  0.6557
    marker-9   80.000  80.000  -0.4877  0.6902  -0.2706  0.6037
   marker-10   90.000  90.000  -0.2924  0.6222  -0.1136  0.5662
```

Your values may be different due to different random seeds in your seed file.

For more details about the `lm_ibdtest` methods, see [Bas08].

See [Concept Index], page 151, for: `lm_ibdtests` sample output.

## 10.4 Sample `civil` parameter file

`civil` bases its tests on the inheritance vectors at the test or conditioning positions. Since these are not observable, a randomized-test strategy is used to deal with this issue. To perform marginal and conditional tests using `civil`, the user must first run the MORGAN program `gl_auto` to draw an MCMC sample of the inheritance vectors jointly at all involved genomic positions: including all possible test positions and conditioning positions. For either the marginal or the conditional test, at each test position, `civil` will compute N test statistic values and N p-values, one for each MCMC realization of inheritance vectors, where N is the size of the MCMC sample. The collection of the N p-values provides an empirical distribution of a randomized (or latent) p-value.

Typically, 5 files are required for running civil, `*.par *.xtra *.ped *.markers *.oscor` and an optional seed file can also be used.

The parameter file `*.par` for `civil` should be based on the one used by `gl_auto` to generate the MCMC realizations of the segregation indicators. It should include MORGAN statements about pedigrees, quantitative traits, markers and sampler seeds. Additional informations on the gl_auto output file and marginal, conditional test setup are specified in an extra parameter file `*.xtra` and provided to `civil` through the '`input extra file`' statement.

For example, in the `civil` parameter file `Autozyg/Gold/civil.vc.par`, the pedigree and marker informations are specified as

```
input pedigree file 'civil.ped'

input marker data file 'civil.markers'
select all markers
```

The pedigree and marker information should be the same as those in the gl_auto par file, except that `civil` requires a quantitative trait to be specified, so a column of quantitative trait values need to be added to the input pedigree file if it is not already there.

In the same par file, a quantitative trait is specified as

```
select trait 2
set trait data quantitative
```

```
input pedigree record trait 2 real 3

set trait 2  tloc 12

set trait 2 for tloc 12 genotype means 0.2000000,  4.9000000,  9.6000000
set trait 2 additive variance 2.0
set trait 2 residual variance 15.0

set tloc 12 allele freqs 0.3 0.7
map test tloc 12 all interval proportions 0.3 0.7
map test tloc 12 external recomb fracts   0.1 0.3 0.45
```

The two '`map test tloc`' statements are required by MORGAN, but the numbers in those lines will not be used by `civil`. The values of '`additive variance`' and '`residual variance`' specified here will be used by `civil` only when '`use_sample_variance`' is set to '`no`' in the extra parameter file (see below). The '`genotype means`' will be used only if '`use_sample_mean`' is set to '`no`' in the extra parameter file.

Additional informations about marginal and conditional test setup are provided to `civil` through an '`extra file`'.

```
input extra file 'civil.vc.xtra'
```

The outline of the extra file is as follows (for an example, see '`Autozyg/Gold/civil.vc.xtra`'):

```
## inheritance vector file name (.oscor file)
civil.oscor
## output file directory
.
## output file keyword
civil
## info on the oscor file ...
n_mcmc  10
order 0
## trait model parameters ...
pD 0.3
use_sample_mean yes
mu 0
use_sample_sd yes
## marginal test parameters
test_statistic vc
n_mc 9999
n_pos 101
test_pos 0 4 8 12 ...
## conditional test parameters
test_statistic vc
n_mc 999
n_pos 81
test_pos 40 44 48 ...
test_pos_l 0 4 8 ...
```

```
          test_pos_r 80 84 88 ...
```

The first 6 lines provide the name of the `gl_auto` output file (line 2), the name of the output directory (line 4), and a keyword for naming the output files (line 6). `civil` will create four output files, suffixed by `*.miv.p.out`, `*.miv.t.out`, `*.civ.p.out`, and `*.civ.t.out`, in the output directory. The four files store marginal and conditional test statistic values and p-values.

The section following '`## info on the oscor file ...`' specifies the number of MCMC scans in the `gl_auto` output file and whether the output is arranged by component or not, with 1 meaning yes and 0 no. If the lines in the s`gl_auto` output is arranged by component, the lines will be rearranged so that they are ordered by MCMC scan and a new file will be created to store the rearranged output file.

The section following '`## trait model parameters ...`' specifies the rare allele frequency of the putative causal variant and specifies how to estimate mean trait value and residual standard error for the trait values: if '`use_sample_mean yes`', then `civil` will use the raw sample mean to estimate the mean trait value, otherwise the mean value specified in the next line will be used. If '`use_sample_sd yes`', then `civil` will use the sample sd to estimate residual standard error, otherwise residual standard error will be estimated by sqrt(residual variance + additive variance) using values provided in the main `civil` parameter file.

The section following '`## marginal test parameters`' specifies the test statistic, the number of Monte Carlo runs for simulating the null distribution (not to be confused with the count of MCMC realizations in the `gl_auto` output scores file), the number of tests requested and the indices to the test positions for the marginal tests. Currently, two test statistic options '`vc`' and '`w`' are available. In this example par file, we ask `civil` to perform 101 marginal tests at positions indexed by 0, 4, 8, ..., 404.

The section following '`## conditional test parameters`' specifies the test statistic, the number of Monte Carlo runs for simulating the null distribution, the number of tests requested, indices to the test positions, indices to the left and right conditioning positions (one line for each set of positions) for conditional tests. In this example par file, we ask `civil` to perform 81 condition tests. The first conditional test will be at position indexed by 40 and be conditioned on positions 0 and 80.

Note that the test positions have to be a subset of marker positions. The idea is to run `gl_auto` using a set of dense markers that should include all potential test and conditioning positions, although not necessarily all markers in the marker data file. When performing marginal and conditional tests, less dense marker positions can be used.

Currently, this extra file has rigid format requirement. Comment lines (starting with ##) can be modified, but no line should be deleted or added, nor should existing lines be broken into multiple lines. The example xtra file `Gold/civil.vc.xtra` can be used as a template for creating new xtra file.

See [Concept Index], page 151, for: sample parameter file for `civil`, latent p-values, randomized p-values.

## 10.5 Sample `civil` output

Since `civil` is still a beta-test program, it does not have an example in the `MORGAN_Examples` directory. Instead, reference is made to the gold standard examples in the main MORGAN source directory, in the subdirectory `Autozyg/Gold`.

Before running the program `civil`, the user needs to run `gl_auto` to obtain an MCMC sample of whole chromosome realizations of meiosis indicators. See Section 9.5 [Running gl_auto example and sample output], page 71, for details. Under the directory `Autozyg/Gold`, the output file `civil.oscor` from a previous `gl_auto` run is provided for demonstration and testing purpose.

Before running `civil`, an output subdirectory must exist. If `vc` is specified as the test statistic, create a subdirectory named `vc` for storing temporary files in the user specified output file directory; if `w` is specified as the test statistic, create a subdirectory named `w`.

To run the example in `Autozyg/Gold` make sure the following files are present there: `civil.vc.par, civil.vc.xtra, civil.ped, civil.markers, civil.oscor`.

In the `Autozyg/Gold` directory, run `civil` by typing

```
          ../civil civil.vc.par > civil.vc.out
```

Information on the progress of the program will be printed to `stdout`, together with summary information about the pedigrees, markers, trait values, and marginal and conditional test setup. For a large number of pedigrees, `civil` can take several hours to finish. Once the program is finished, four output files, `*.miv.?.t.out, *.miv.?.p.out, *.civ.?.t.out, *.civ.?.p.out`, will be written to the specified output file directory: '`*`' is the output file keyword specified in the xtra file and '`?`' is the name of the specified test statistic ('`w`' or '`vc`'). They store marginal test statistic values, marginal test p-values, conditional test statistic values, conditional test p-values.

The upper left portion of a marginal test p-values file `Autozyg/Gold/civil.miv.m.p.out` is shown below:

```
    test_pos  test_map       pval0          pval1          pval2          ...
    0        0.000000       0.214400       0.098700       0.357800       ...
    4        1.000000       0.305700       0.108900       0.142800       ...
    8        2.000000       0.327400       0.133200       0.132700       ...
    ...
```

In this output file, the first row is the header. Each of the remaining rows corresponds to one marginal test. The first two columns are the index and the map position of the test position. The columns 3 to N + 2 are the test p-values, one for each MCMC realization of the meiosis indicators. The layout of the marginal test statistic file is similar.

The conditional test p-values file '`Autozyg/Gold/civil.civ.m.p.out`' has more columns. For each test, the first 6 columns now correspond to indices to conditional test position, left conditioning position and right conditioning position; then map positions of the conditional test position, left conditioning position and right conditioning position. Starting from column 7 are the N p-values, one for each MCMC realization.

Many temporary files will also be created under the subdirectories '`vc`' or '`w`' of the output directory. These files store intermediate results for computing the test scores. These results will be reused to save time when more tests need to be performed: for example, the user may want to perform more marginal and conditional tests at different test or conditioning positions.

However, if pedigree structures or trait values in the pedigree file, or trait parameters in the '`extra file`' file have changed since last run, these temporary files should not be reused and should be deleted before running `civil`. If pedigree structures have changed, `gl_auto` also need to be rerun. Use the `overwrite` option for the `gl_auto` output scores file, to overwrite the previous file, and/or rename the previous file if you wish to retain it.

See [Concept Index], page 151, for: `civil` sample output.

## 10.6 `lm_ibdtests` and `civil` statements

The programs `lm_ibdtests` and `civil` use the pedigree, and genetic map and marker statements of previous sections.

- For the MCMC statements used by `lm_ibdtests` see Section 8.6 [MCMC parameter statements], page 61.
- For the `gl_auto` statements used by `civil` see Section 9.8 [Autozyg statements], page 74.

The following statements are specific to `lm_ibdtests`:

`compute (ibd | likelihood-ratio) statistics`

Required: one of the two options must be specified.

`output (sampler | permutation) seeds only`

The program `lm_ibdtests` uses random seeds for its permutation testing in addition to the usual MCMC sampler seeds. If an output seed file is named, both ending permutation and sampler seeds will be saved unless only one or the other is requested.

`set ibd measures [Spairs] [Srobdom] [Saffect] [Slambda]`

Optional. `lm_ibdtests` uses 1 to 4 measures to perform *ibd* tests for linage; these are specified in the order [Spairs] [Srobdom] [Saffect] [Slambda]. Spairs, Srobdom, and Slambda may be specified for both normal and permutation tests; Saffect may not currently be specified with the normal tests option.

`set ibd tests [normal] [permutation]`

Optional. Normal and/or permutation tests may be specified.

`set ibd permutations` *I*

Optional. Need to be specified when the permutation test is requested through '`set ibd tests`'. The default is 999. It is recommended that at least 50 permutations are used.

`set likelihood-ratio lambda-p model gridpoints` *I1 I2*

When the lambda_p measure is used for the chi-square likelihood-ratio test), the number of gridpoints may be specified. The number *I1* is the number of gridpoints in the interval for the lambda-parameters of the model, and *I2* is the number of gridpoints in the interval for p. The default is 6 and 9, respectively.

`set likelihood-ratio measures [delta] [lambda_p]`

When computing the chi-square likelihood-ratio test, the choice of measures is delta and/or lambda_p, in the order [delta] [lambda_p]. The default is '`delta`'.

`set likelihood-ratio tests`

When computing likelihood-ratio statistics, chi-squared tests are performed. Thus, this statement is presently redundant, as there is no choice in tests.

`set permutation seeds` *`H1 H2`*
: The program `lm_ibdtests` uses random seeds for its permutation testing in addition to the usual MCMC sampler seeds. The seeds may be specified in the `input seed file` or in the parameter file: otherwise default seeds will be used.

The program `civil` has no program-specific parameter statements. Instead information is provided to `civil` using the `input extra file statement`:

`input extra file` *`filename`*
: Required

For information about the contents of the extra file see Section 10.4 [Sample civil parameter file], page 83.

See [Concept Index], page 151, for: `lm_ibdtests` statements, `civil` statements, *ibd* measures, likelihood-ratio measures.

# 11 Estimating Location lod Scores by MCMC

See [Concept Index], page 151, for: location lod scores estimates.

## 11.1 Introduction to lm_linkage, lm_bayes, lm_twoqtl, gl_lods and base_trait_lods

The programs lm_linkage, lm_bayes, gl_lods and lm_twoqtl are referred to as Lodscore programs. The program lm_linkage replaces the two programs lm_markers and lm_multiple of pre-2011 versions of MORGAN. As of 2013, the program lm_twoqtl remains a beta test version, but the new program gl_lods is now fully implemented and documented.

The Lodscore programs use MCMC to perform multipoint linkage analysis and trait mapping on large pedigrees where many individuals may be unobserved and exact computation is infeasible. The data are the genotypes of observed individuals in the pedigree at marker loci and discrete or continuous trait data. As with exact methods of computing lod scores, the genetic model is assumed known. The only unknown parameter is the location of the trait locus. Therefore, the user is required to specify the marker locations, trait and marker allele frequencies and penetrance function. Presently, users are limited in their choice of penetrance function, but this is under revision and will change in future releases of MORGAN.

lm_linkage is an implementation of the Lange-Sobel estimator, using either the single- or multiple-meiosis LM-sampler: See Section 8.4 [Single and multiple meiosis LM-samplers], page 58. The Lange-Sobel estimate works reasonably well in reasonable time, provided a good MCMC sampler is used, and provided the trait data do not have strong impact on the conditional distribution of meiosis indicators. The lm_linkage program samples only the meiosis indicators at marker loci, and only conditional on the marker data. Even when the trait inheritance information is strong, the method can produce quite accurate lod scores in the absence of linkage, but it can be inaccurate in estimating the strength of linkage signals. As well as producing the lod score, our current implementation provides a batch-means pointwise estimate of the Monte Carlo standard error of the lod-score estimate. lm_linkage can work with genotypic, discrete or quantitative traits.

lm_linkage combines the earlier programs lm_markers and lm_multiple. The original lm_multiple program and multiple-meiosis sampler are the work of Liping Tong [TT08]. As well as allowing use of either the single- or multiple-meiosis LM-sampler, the lm_linkage program optionally perform exact lodscore computations on small pedigree components, and includes better exact computation and pedigree peeling options for use in the lod score estimator (see Section 8.3 [Exact HMM computations], page 57).

lm_bayes is an alternative method implemented for genotypic or discrete traits. The MCMC performance is better than for the old lm_markers program, but it has other computational overheads. lm_bayes samples trait locations from a posterior distribution, and then divides it by the prior to produce the likelihood and hence the lod score. Estimation is in two phases. A preliminary run with discrete uniform prior gives order-of-magnitude relative likelihoods. Then, using the inverse of these likelihoods as prior weights of a 'pseudo-prior' distribution. Using this 'pseudo-prior' a second run is made to estimate the likelihood. The purpose of the 'pseudo-prior' is to produce an approximately uniform posterior, so that likelihoods will be well estimated at all test positions. It is important that the initial run is long enough for all test positions to be sampled, and for the unlinked trait position to have a reasonable number of realizations. For locations at which lod scores are very negative, or for the unlinked position when there is some linked location with strong positive lod score, this can be problematic.

Our current implementation of `lm_bayes` provides two lod score estimates. The first is a crude estimate which counts realizations of locations sampled to estimate the posterior: as can be seen from the output this can be quite erratic. The Rao-Blackwellized estimator is much preferred, and produces good estimates in reasonable time. The `lm_bayes` program is the work of Andrew George [GT03,GWT05].

The beta-test program `lm_twoqtl` does parametric linkage analysis for a quantitative trait model having one or two linked QTL and a polygenic component. Each QTL has two alleles with three different genotypic means. The Normally distributed polygenic component does not include dominance, and the environmental contribution is has a Normal distribution with mean zero and uncorrelated among individuals. The program output consists of MCMC-based lod score estimates of the joint locations of the one or two contributing QTL. As of 2011, the program uses the same MCMC options as `lm_linkage` for sampling descent at marker loci conditional on marker data. Conditionally on these realizations the program then uses exact computation (on very small pedigrees) or an additional level of Monte Carlo to estimate the relevant lod score contributions. The original versions of the `lm_twoqtl` were the work on YunJu Sung [STW07,SW07].

The program `gl_lods` computes lod score contributions for a discrete or continuous trait given a set of ibd_graphs across the chromosome, produced by `gl_auto`: See Section 9.1 [Introduction to lm_auto gl_auto and lm_pval], page 63. If the `gl_auto` run uses the '`set MCMC markers only`' option, then the overall lod score computed by `gl_lods` is identical to that produces by `lm_linkage` when the same MCMC options are used in the in `gl_auto` and in `lm_linkage`. `gl_lods` many of the same trait definition and mapping request parameter statements as `lm_linkage` (Section 11.9 [Location lod scores statements], page 111). However its input consists of ibd_graphs and an individuals file; there are no marker data or pedigree data. For further information on the motivation for splitting of the `lm_linkage` lod score computation into the generation of marker-based ibd graphs (using `gl_auto`) followed by trait-likelihood computation on the ibd graphs: See Section 11.6 [Parameter files for the gl_lods program], page 101. See also [Tho11].

Because the `gl_lods` program computes log-likelihood contributions for given IBD, and does not use a pedigree, there is no way to compute the usual normalizing factor of the pedigree-based probability of observed trait data. This must be supplied to the program, and may be computed via the `base_trait_lods` program, provided a pedigree and the same trait model and data are used. An alternative approach using permutation of trait data conditional on the locus-specific IBD was developed by Chris Glazner [GT15] and has been implemented as an option in `gl_lods`. For quantitative data using variance component models, we are investigating the used of the Kullback Leibler information (expected lod score) conditional on the locus-specific IBD as a locus-specific normalizing factor.

See [References], page 149, for details of the cited papers.

See [Concept Index], page 151, for: Markov chain Monte Carlo, `lm_linkage` introduction, `lm_bayes` introduction, `lm_twoqtl` introduction, meiosis indicators, multiple meiosis sampler.

## 11.2 Sample parameter files for `lm_linkage` and `lm_bayes`

There are three example parameter files in the `Lodscores` subdirectory: `ped73_ge.par`, `ped73_ph.par` and `ped73_qu.par`. These files are examples of how to analyze genotypic, discrete (phenotypic), and quantitative (continuous) traits, respectively. Each of these files is written for use with `lm_linkage` since this is our preferred program and can analyze genotypic, discrete, and quantitative traits. The program `lm_bayes` will run with the same parameter files `ped73_ge.par` and `ped73_ph.par`, but will adopt defaults for several statements specific to this program and will generate warning for others not implemented for `lm_bayes`. If `lm_bayes` is run using `ped73_qu.par`, all statements regarding quantitative traits will be ignored, and the program will use default genotypic data.

The marker and MCMC information is very similar for all three parameter files. For `ped73_qu.par` it is as follows:

```
set printlevel 5

input pedigree file '../ped73.ped'
input marker data file '../ped73.marker.missing'
input seed file         '../sampler.seed'
output overwrite seed file        '../sampler.seed'

set trait 1 data quantitative
input pedigree record trait 1 real 1

select all markers
select trait 1
set trait 1 tloc 1
map test tloc 1 all interval proportions 0.3 0.7
map test tloc 1 external recomb fracts   0.05 0.15 0.3 0.4 0.45

sample by scan
set L-sampler probability 0.2

set burn-in iterations 150
set MC iterations 3000
check progress MC iterations 1000

set global MCMC
use single meiosis sampler
```

The pedigree file specified by the '`input pedigree file`' statement can contain multiple traits. As discussed in previous sections, the marker map, allele frequencies and genotypes can be contained in the parameter file or in a separate file specified by the '`input marker data file`' statement as in the example above.

As in other programs, the trait data are included in the pedigree file. The '`select trait`' statement tells the program which trait in this file is to be analyzed, and the '`input pedigree record trait`' indicates where the data are to be found, while the '`set trait ...tloc...`' statement connects the trait with a specific tloc for this analysis.

The two '`map test tloc`' statements give trait locus test positions at which the lod scores should be calculated. When the trait locus is located between two markers, the position is specified in terms of the proportional genetic distance between the two markers (this option makes handling gender-specific maps easy). In this example, the test trait positions are specified to be at 30 and 70 percent of the interval. The second '`map test tloc`' statement allows test trait locus positions located before the first marker or after the last marker to be specified; the positions are specified explicitly in terms of recombination fractions (or genetic distances) with the nearest marker locus. Note that an external recombination fraction of 0.5 is not necessary since the likelihood of an unlinked trait locus is always used as a reference when computing the lod scores.

The final seven statements give MCMC specifications. The '`sample by scan`' statement instructs the program to update all the meiosis indicators, **S**, at each iteration, in an order determined by random permutation. The alternative '`sample by step`' updates only one locus (L-sampler) or only one meiosis (M-sampler) in each iteration. The '`sample by scan`' statement is the default and strongly recommended. The L-sampler probability is set at 20 percent, which is often a good choice. For a detailed discussion of effects of varying L- to M-sampler ratio, see section 10.6 in [Tho00].

In the '`set burn-in iterations`' statement, 150 burn-in iterations, are requested. The next statement requests 3000 MCMC iterations; for each realized set of marker-location inheritance vectors the trait-likelihood contribution will be computed at each test position of the trait locus. This is for demonstration purposes only. For real data analyses, use longer runs, on the order of 10^5 MCMC iterations. The last statement in this group tells the program to report progress every 1000 iterations.

Although the `lm_linkage` program can use the multiple-meiosis sampler, and this is recommended, the final two statements here specify '`set global MCMC`' and '`use single meiosis sampler`'. Thus, for this example, the single-meiosis sampler will be used (as in the old `lm_markers` program) and MCMC will be performed globally over all pedigree components, rather than component-by-component. This provides an example of how these options may be used for compatibility with older examples.

As an example of MORGAN parameter statement flexibility we give also a part of a version of the above parameter file in which lines are broken, words mis-spelled, and lower/upper case used arbitrarily::

```
#  This version of ped73_qu.par is designed to show the flexibility
#     of MORGAN parameter statements

set prinlev 5

inPU Pedi file '../ped73.ped'
input mark
data FILE '../ped73.marker.missing'
input seed file         '../sampler.seed'
outp over
seed file        '../sampler.seed'

set trait data 1 quant
Input pedigree Recod trait 1 real 1
```

```
sele traix 1
set trait 1
tloc 1
map test tloc 1 all intevl props 0.3 0.7
map test tloc 1 exte reco frac 0.05 0.15
0.3 0.4 0.45
seLect
all markers

samp BY scan
set L-sam prob 0.2
set 3000 mc ITER
set burn-in iter 150
check progss 1000 MC iters
set glob mcmc
USE singe meio samp
```

Note that statements may be split over lines, that only the first four characters of each word is required, that upper and lower case are not distinguished, and that in some statements (for example, the '`check progress`' and '`set MC iterations`' statements) even the location of the integer count within the statement is flexible.

For more details of the MCMC specifications see Section 8.6 [MCMC parameter statements], page 61.

**Specifying Trait Data Type**

Trait data type is set by using the '`set trait data`' statement. Recall that the '`input pedigree record trait`' statement must be used to specify which column in the file is to be used as the trait value (see Section 2.9 [Pedigree file description statements], page 15). The three trait data types discussed in this example are implemented by including the following statements in the parameter file discussed above. Note the trait and numbers are arbitrary, but the connection must be made consistently through the file.

`ped73_ge.par` specifies a genotypic trait with the following statements:

```
set trait 3 data genotypic
input pedigree record trait 3 integer 3

select trait 3
set trait 3 tloc 1
set tloc 1 allele freqs 0.4  0.6
```

`ped73_ph.par` specifies a phenotypic trait with the following statements:

```
set trait 2 data discrete
input pedigree record trait 2 integer 4

select trait 2
set traits 2 tlocs 1
set traits 2 for tlocs 1 incomplete penetrance 0.05 0.6 0.95
set tlocs 1  allele freqs 0.4  0.6
```

Recall that for discrete data, one must specify the penetrances (see Section 9.8.7 [Autozyg computational parameters], page 77).

`ped73_qu.par` specifies a quantitative trait with the following statements:

```
set trait 1 data quantitative
input pedigree record trait 1 real 1

select trait 1
set trait 1 tloc 1
set trait 1 for tlocs 1 genotype  mean 90.0 100.0 110.0
set trait 1 residual variance 25.0
set tloc 1 allele freqs 0.4 0.6
```

When using a quantitative trait, genotypic means and residual variance must be specified. Additive variance can be specified with the statement '`set trait ... additive variance`'. The default value is zero.

The '`set tloc ... allele freqs`' statement specifies allele frequencies at the trait locus. If the allele frequencies sum to less than 1, a warning message will be issued:

```
Sum of allele frequencies is not in range .9999, 1.0001  (W)
```

If the allele frequencies sum to above 1.0001, the program quits and generates an error message.

Below is a summary of the trait data types accepted for each program:

| | **Genotypic** | **Phenotypic** | **Quantitative** |
|---|---|---|---|
| | *ped73_ge.par* | *ped73_ph.par* | *ped73_qu.par* |
| **lm_linkage** | Yes | Yes | Yes |
| **lm_bayes** | Yes | Yes | No |

Additional penetrance options are available for liability classes and age-of-onset data.

An example is given in the file `xact_ph_liab.par` in the `Lodscore` Gold standards. The statement gives the liability class specific penetrance matrix. The penetrances are for the 11, 12, 22 genotypes. Morgan will set the penetrances for genotype 21 equal to that for 12.

```
set trait data 1 discrete with liability
input pedigree record trait 1 integer pairs 8 11

set 3 liability classes with penetrances
        0.025 0.325 0.325
        0.150 0.625 0.625
        0.350 0.950 0.950
```

An example is given in the file `xact_ph_age.par` in the `Lodscore` Gold standards:

```
input pedigree record trait 1 integer pairs 8 10  # For using ages of onset.
set trait 1 data discrete with covariate

set trait 1 for tloc 1 genotypic means  90.0 70.0 45.5
set standard deviations for genotypes   11.0 15.5 20.5
```

```
    set width of ages of onset window  2.0
```

See [Concept Index], page 151, for: sample parameter file for `lm_linkage`, sample parameter file for `lm_bayes`, gender–specific maps, meiosis indicators, L-sampler, M-sampler, multiple meiosis sampler, genotypic trait specification, phenotypic trait specification, discrete trait specification, quantitative trait specification, continuous trait specification, liability class penetrances, age-of-onset penetrances.

## 11.3 Running `lm_linkage` examples and sample output

`lm_linkage` can be run with all three parameter files in the `Lodscores/` subdirectory. As usual, the syntax for running the program is:

```
    ./lm_linkage <parameter file>
```

This section describes the output obtained by using the parameter file `ped73_qu.par`. To run the example, type:

```
    ./lm_linkage ped73_qu.par
```

The interesting part of the output is the LodScore estimates. For each test position, we have the estimated lod score and the estimated Monte Carlo standard error.

```
    LodScore estimates by Rao-Blackwellized computation:

     Trait pos #     position (Haldane cM)
       or marker         male     female       LodScore    StdErr

               1    -115.129   -115.129         0.0303    0.0005
               2     -80.472    -80.472         0.0558    0.0012
               3     -45.815    -45.815         0.0779    0.0031
               4     -17.834    -17.834        -0.0306    0.0080
               5      -5.268     -5.268        -0.2811    0.0142
        marker-1       0.000      0.000        -0.4986    0.0195
               6       3.000      3.000        -0.4469    0.0141
               7       7.000      7.000        -0.4342    0.0230
        marker-2      10.000     10.000        -0.4605    0.0363
               8      13.000     13.000        -0.4254    0.0247
               9      17.000     17.000        -0.4454    0.0209
        marker-3      20.000     20.000        -0.5301    0.0197
              10      23.000     23.000        -0.3174    0.0211
              11      27.000     27.000        -0.1176    0.0233
        marker-4      30.000     30.000        -0.0052    0.0259
              12      33.000     33.000         0.5058    0.0208
              13      37.000     37.000         0.8794    0.0159
        marker-5      40.000     40.000         1.0772    0.0138
              14      43.000     43.000         0.9832    0.0156
              15      47.000     47.000         0.8432    0.0213
        marker-6      50.000     50.000         0.7210    0.0252
              16      53.000     53.000         0.6558    0.0256
              17      57.000     57.000         0.5140    0.0271
        marker-7      60.000     60.000         0.3522    0.0288
```

```
         18      63.000     63.000           0.0113    0.0225
         19      67.000     67.000          -0.5473    0.0123
   marker-8      70.000     70.000          -0.9543    0.0095
         20      73.000     73.000          -0.4578    0.0212
         21      77.000     77.000          -0.1866    0.0178
   marker-9      80.000     80.000          -0.1135    0.0116
         22      83.000     83.000           0.0888    0.0091
         23      87.000     87.000           0.3132    0.0064
  marker-10      90.000     90.000           0.4544    0.0071
         24      95.268     95.268           0.6010    0.0046
         25     107.834    107.834           0.6423    0.0028
         26     135.815    135.815           0.4017    0.0011
         27     170.472    170.472           0.1762    0.0003
         28     205.129    205.129           0.0758    0.0001
```

For more information regarding the MCMC parameters and diagnostic output, See Section 8.5 [MCMC computational options], page 59.

See [Concept Index], page 151, for: running `lm_linkage` examples, `lm_linkage` sample output.

## 11.4 Running `lm_bayes` examples and sample output

Under the subdirectory `Lodscores/`, run the `lm_bayes` example on the discrete (phenotypic) trait data by typing:

```
./lm_bayes ped73_ph.par
```

The results from `lm_bayes` are the lod scores toward the end of the output. Two estimates of the lod scores are provided: (1) count realizations of locations sampled to estimate the posterior probability (‘`crude`’) and (2) Rao-Blackwellized estimator (‘`R-B`’). Both are provided for comparison, but the latter should be more accurate.

```
LodScore estimates:

Trait pos #  position (Haldane cM)  pseudo  freq       LodScore
  or marker      male    female     prior  visited   crude     R-B

          0  unlinked  unlinked  0.025023    94        NA        NA
          1  -115.129  -115.129  0.025276    66   -0.1580   -0.0046
          2   -80.472   -80.472  0.025727    77   -0.0987   -0.0125
          3   -45.815   -45.815  0.027843    96   -0.0372   -0.0473
          4   -17.834   -17.834  0.037973    71   -0.3030   -0.1825
          5    -5.268    -5.268  0.057289    96   -0.3506   -0.3583
   marker-1     0.000     0.000        NA    NA        NA        NA
          6     3.000     3.000  0.078826    89   -0.5221   -0.4919
          7     7.000     7.000  0.086379    88   -0.5667   -0.5255
   marker-2    10.000    10.000        NA    NA        NA        NA
          8    13.000    13.000  0.092502    87   -0.6014   -0.5456
          9    17.000    17.000  0.090858    94   -0.5600   -0.5386
   marker-3    20.000    20.000         NA   NA       NA       NA
         10    23.000    23.000   0.063483  109  -0.3400  -0.3738
         11    27.000    27.000   0.044111  103  -0.2065  -0.2086
   marker-4    30.000    30.000         NA   NA       NA       NA
         12    33.000    33.000   0.026053  114   0.0663   0.0203
         13    37.000    37.000   0.018403  103   0.1731   0.1698
   marker-5    40.000    40.000         NA   NA       NA       NA
         14    43.000    43.000   0.011818  100   0.3527   0.3585
         15    47.000    47.000   0.009347   90   0.4088   0.4600
   marker-6    50.000    50.000         NA   NA       NA       NA
         16    53.000    53.000   0.010351  121   0.4930   0.4236
         17    57.000    57.000   0.014614  121   0.3432   0.2804
   marker-7    60.000    60.000         NA   NA       NA       NA
         18    63.000    63.000   0.023348   96   0.0392   0.0769
         19    67.000    67.000   0.030506  123   0.0307  -0.0412
   marker-8    70.000    70.000         NA   NA       NA       NA
         20    73.000    73.000   0.033357  136   0.0356  -0.0903
         21    77.000    77.000   0.030400  124   0.0358  -0.0514
   marker-9    80.000    80.000         NA   NA       NA       NA
         22    83.000    83.000   0.024811   96   0.0128   0.0282
         23    87.000    87.000   0.019535  144   0.2928   0.1160
  marker-10    90.000    90.000         NA   NA       NA       NA
         24    95.268    95.268   0.013755  110   0.3281   0.2561
         25   107.834   107.834   0.013714  125   0.3849   0.2600
         26   135.815   135.815   0.018372  132   0.2816   0.1339
         27   170.472   170.472   0.022361  108   0.1091   0.0489
         28   205.129   205.129   0.023966   87  -0.0149   0.0188
```

Note that `lm_bayes` does not provide lod scores at the marker locations.

See [Concept Index], page 151, for: running `lm_bayes` examples, `lm_bayes` sample output, Rao-Blackwellized estimates.

## 11.5 Running `lm_twoqtl` examples and sample output

The program `lm_twoqtl` remains beta test, so that instead of examples in `MORGAN_Examples` we describe the gold standard example in the main MORGAN source directory in subdirectory `Lodscore/Gold`. However, much work has been done on `lm_twoqtl`, so that its marker-based MCMC is now as for `lm_linkage`. Four gold-standard examples of the `lm_twoqtl` parameter files and output may be found in the '`Lodscore/Gold`' directory of MORGAN. Examples 1 and 3 are for a single trait locus, and 2 and 4 use two QTL. Examples 1 and 2 use exact computation for the trait lod score contributions on these very small examples. Examples 3 and 4 use Monte Carlo. We will use example 4 in this tutorial description, since this is the most general and novel. To create other examples, copy one of these files and replace the parameters in the file with those that you want to specify.

The various trait-model options for `lm_twoqtl` are summarized in the following table:

| Additive Genetic Variance: | | Zero | Positive |
|---|---|---|---|
| Number of QTL: | 1 | one locus | one locus plus polygene |

| | | |
|---|---|---|
| 2 | two loci | two loci plus polygene |

Trait models can be any of the above four entries. However, for a one-locus trait model with no polygenic component, the program `lm_linkage` will provide more accurate results more quickly.

The lod score is estimated on a one-dimensional grid of points for one QTL, and a two-dimensional grid of points for two QTL. In the future the new parameter statement

```
map [chromosome  I]  test tlocs  L1 L2  jointly at markers J11 J12 ...
```

will allow two-locus lod score programs that provide lod scores at arbitrary pairs of marker positions.

The content of file `twoqtl4.par` (reordered slightly for clarity) is:

```
use single meiosis sampler           # Select the MCMC sampler to be used.

set printlevel 5                     # Include everything in the output file.

set sampler seeds  0x53f78285 0xdfbca001
set trait   seeds  0x53f78285 0xdfbca001

input marker data file ‘./twoqtl.markers’
input pedigree file ‘./twoqtl.ped’
output extra file ‘./twoqtl_batch4’

select all markers
select trait 1

set trait 1 multiple tlocs 1 2
set tloc 1  allele freqs 0.1 0.9
set tloc 2  allele freqs 0.3 0.7

# requests for grid of tloc positions for lod scores
map test tloc 1 all interval proportions 0.5
map test tloc 1 external recomb fracts   0.3
map test tloc 2 all interval proportions 0.5
map test tloc 2 no default external positions

# standard MCMC requests
use sequential imputation for setup
use 100 sequential imputation realizations for setup
sample by scan
set L-sampler probability 0.2
set burn-in iterations 10
set MC iterations 60
compute scores every 10 iterations

# lodscore scoring requests
set 3 batches MC variance estimation
```

```
    check progress 20 MC iterations
    # For clarity, the wording 'MC' has been removed from the following three  parameter s
    #  MC in parameter statements now refers only to MCMC.
    use Monte Carlo summation for trait
    simulate 5 IBD realizations for trait
    use multiplier 1 realization for null

    # quantitative trait model specification
    set trait 1 data quantitative
    set trait 1 for tloc 1 genotype  mean  -2.0  0.0  2.0
    set trait 1 for tloc 2 genotype  mean  -3.0  0.0  3.0
    set trait 1 residual  variance  1.0
    set trait 1 additive  variance  1.0
```

Note the number of MCMC scans (60) is very small, as also are the number of Monte Carlo realizations to be used in evaluating the trait likelihood contributions (5). Additionally, only every 10th MCMC scan is used for computing lod score contributions. This is reasonable, in that for `lm_twoqtl` lod score computation is computationally intensive, so that the standard procedure of scoring every scan is not efficient. However, with only 60 total scans, this means lod scores are based on only 6 realizations of inheritance conditional on the marker data. The example is for illustrative purposes only; in real examples much more Monte Carlo would be required both in the marker-based MCMC and for estimating trait contributions to each score.

Most statements are as for earlier lod-score programs and can be found in the [Statement Index], page 155. The statements included in this example that require additional comment are

`use single meiosis sampler`
: The old single-meiosis sampler is specified for consistency with earlier results; in practice the `multiple meiosis sampler` is preferred.

`set trait 1 multiple tlocs 1 2`
: This statement specifies that trait 1 is contributed to by both tloc 1 and tloc 2.

`set trait 1 for tloc 1 genotype mean -2.0 0.0 2.0`
`set trait 1 for tloc 2 genotype mean -3.0 0.0 3.0`
: The genotypic means for tlocs 1 and are set separately, which will imply their additive contribution to trait 1. If the tlocs are not to contribute additively, the user should instead use the statement

  `set trait 1 for tlocs 1 2 joint genotype means ...` followed by 9 genotype means for the two tloc genotype combinations.

`output extra file ./twoqtl_batch4`
: If an '`extra file`' is specified, it is used by `lm_twoqtl` for output of batched means used in variance estimation. Most users will not require this file, although it can be used in MCMC diagnostics.

`set 3 batches MC variance estimation`
: In this minimal example, the 6 scored realizations are divided into 3 batches, each of size 2. Again, real examples would use much larger number of realizations, and likely the default number of batches (20).

`use Monte Carlo summation for trait`
`simulate 5 IBD realizations for trait`
: In this example, Monte Carlo summation is to be used for evaluating each trait-locus likelihood contributions conditional on marker-based realizations of inheritance, and for each such realizations there will be 5 realizations of trait allele descent.

`use multiplier 1 realization for null`
: In this example the same number of realizations will be used to evaluate the marginal probability of trait data as are used for each lodscore grid point. In real examples, it may be advisable to increase this ratio, to obtain an accurate base level for the lodscore estimate.

The default procedure of estimation of lod scores on each of the two components separately is used. These are then summed, giving the final concluding output for this example:

```
# Lod score estimates and MC sd for entire pedigree:

# Index     TLoc1       TLoc2     LodScore    StdErr

     1    -45.815     -45.815       1.4058    0.7605
     2    -45.815       0.000       0.3872    0.6212
     3    -45.815      10.000       0.4379    0.6232
     4    -45.815      20.000       0.6616    0.6856
     5    -45.815      65.815       0.4661    0.5916
     6      0.000     -45.815       0.3503    0.6529
     7      0.000       0.000       0.1343    0.6034
     8      0.000      10.000      -0.0129    0.5600
     9      0.000      20.000       1.0364    0.5772
    10      0.000      65.815       1.1077    0.7724
    11     10.000     -45.815       0.0983    0.6902
    12     10.000       0.000       0.0461    0.5605
    13     10.000      10.000       0.8585    0.6088
    14     10.000      20.000       1.2174    0.5866
    15     10.000      65.815       0.3512    0.7539
    16     20.000     -45.815       1.5180    0.6479
    17     20.000       0.000       0.7387    0.6567
    18     20.000      10.000       0.9330    0.7020
    19     20.000      20.000       1.2473    0.6162
    20     20.000      65.815       1.3465    0.7664
    21     65.815     -45.815      -0.3066    0.7764
    22     65.815       0.000      -0.1259    0.6895
    23     65.815      10.000       0.5714    0.7124
    24     65.815      20.000       1.3537    0.6550
```

```
        25      65.815      65.815       0.5343    0.7140
```

These results consist of base-10 lodscore estimates with MCMC standard deviations, estimated at the requested grid of test positions.

See [Concept Index], page 151, for: running `lm_twoqtl` examples, `lm_twoqtl` sample output, map test tlocs jointly at markers.

## 11.6 Parameter files for the `gl_lods` program

See [References], page 149, for details of the cited papers.

The program `gl_lods` computes lod score contributions for a discrete or continuous trait given a set of ibd_graphs across the chromosome, produced by `gl_auto`: See Section 9.1 [Introduction to lm_auto gl_auto and lm_pval], page 63. If the `gl_auto` run uses the '`set MCMC markers only`' option, then the overall lod score computed by `gl_lods` is identical to that produced by `lm_linkage` when the same MCMC options are used in the in `gl_auto` and in `lm_linkage`. `gl_lods` many of the same trait definition and mapping request parameter statements as `lm_linkage` (Section 11.9 [Location lod scores statements], page 111). However its input consists of ibd_graphs and an individuals file; there are no marker data or pedigree data.

The goal of using `gl_auto` and `gl_lods` is to separate the lod score computation from the marker-based MCMC that produces realizations of the inheritance vectors at loci across the chromosome. The input to `gl_lods` consists of these realizations, in the format of `gl_auto` output compressed ibd_graphs, a specification of a trait model, and a list of individuals with their trait data. `gl_lods` computes lod score contributions for a specified trait model and data on specified individuals, directly using the *ibd* graphs on these individuals. See also [Tho11]. From MORGAN 3.2. the `gl_lods` program is no longer beta-test; parameter statements, capabilities, and examples have been significantly updated.

The separation of lod-score computation and marker-based MCMC has several advantages:

- Lod score contributions are computed for each ibd graph, providing a distribution ("fuzzy lod") over the ibd-graph realizations.
- Use of IBDgraph software to identify equivalent ibd graphs across realizations and across markers, will enable lod score contributions to be computed once only for each distinct ibd graph;
- Lod scores for many different genetic models and for different traits may be computed on the same set of ibd graphs.
- The pedigree and marker information may be separated from the trait data. This increases data confidentiality in multi-center genetic studies.

Examples for `gl_lods` are still under development, but we describe here two examples based on the `gl_lods` gold standards in the `Lodscore/Gold` subdirectory of MORGAN. In the case of the first `ped47_...` example the files are only in that directory and can be run there as '`make gold.6`'. The other example `simCmps_fgl_...` is both included as a gold standard, and has also been added to the `MORGAN_Examples` files.

- **The ped47 example:** The `ped47` example is a single pedigree component. The following shows the parameter file `ped47_gl_lods.par`. The trait model specifications for the two cases of a discrete and a continuous trait are given in supplementary parameter files below.

```
# ped47 main parameter file for gl_lods program.

set printlevel 5            # Include everything in the output file.

input individuals file        "./ped47.ind"
input extra file              "./ped47_fgl.oscor"
output overwrite extra file  "./ped47_fgl.reduce"
# Use the overwrite option to avoid appending multiple outputs.

# The number of ibdgraphs provides the number of replicates in gl_auto file.
set maximum 1000 ibdgraphs per component

# This map test tloc statement indicates at which markers lodscores are to be
# calculated.
map test tloc 11 at markers 2 4 5 8 10

select trait 1
set trait 1 tloc 11
set tloc 11 allele freqs 0.5 0.5
```

Note that the program uses an individuals file, not a pedigree file. The format is similar, but mother and father identifiers are replaced by a component number (in this case '1' for all individuals). See Section 3.4 [Creating an individuals file], page 20. Note that the component number is referred to as a "name", so that the columns of integers and reals in the file correspond to those in an analogous pedigree file. The individuals file `ped47.ind` is as follows:

```
# Lines or parts of lines preceded by a hash sign, such as this one,
#    are treated as comments by the software.
# This individuals file contains a subset of the 47 individuals in the gl_auto
#   output file; the subset must include all those whose trait data are
#   to be analyzed -- it may include additional individuals.

input pedigree size 42
#  pedigree size must be the number of individuals in the fgl-graphs of the
#    gl_auto output file.
#  The number read here may be only a subset; if so make sure file ends
#    with the last individual.

input individuals record names 2 integers 7 reals 1

#  The first item is individual's "name" which is a character string.
#     The second of these is individual's component.
#
```

```
#  There follow 7 integers
#     The first of these (third item) is dummy gender (0)
#     The next is an "observed" indicator used only by genedrop.
#     The next is a trait genotype:
#          0 is unobserved. 1 is genotype (1,1); 3 is (1,2), 4 is (2,2)
#     The next is a trait phenotype:
#          0 is unobserved. 1 is unaffected, 2 is affected
#     The next 2 code trait-locus inheritance patterns, used by 1 version of
#          markerdrop.
#     The final is a dummy trait indicating data for reduced fgl file
#         (but it is not required/used by the gl_lods program)
#
#  Finally there is one real (read as double); this is a quantitative trait.
#     Numbers with integer part 999 code for unobserved.
#
#  In fact, many individuals with no data are dropped;
#  This individuals  file has only 35 individuals.
#  Since 35<42 (the requested pedigree size), all will be read).
**************************************************
302   1  1  1 3 2  1  1 1  105.945
306   1  2  1 3 1  1  1 1  99.822
307   1  0  1 4 2  1  1 1  111.696
308   1  0  0 0 0  1  1 0  999.5
<lines omitted here>
5080  1  0  0 0 0 -1 -1 0  999.5
601   1  0  1 4 2  1  1 1  112.285
5160  1  0  1 3 1 -1 -1 0  97.043
5150  1  0  1 3 1 -1 -1 1  97.043
602   1  0  1 4 2  0  1 1  105.991
```

Note that the input/output extra file is used to process the file of IBD graphs that were produced (normally by `gl_auto`) in the analysis of marker data. In this example the file is a `gl_auto` output scores file, input here as an `extra` file: `ped47_fgl.oscor`. This file contains 100 ibd graphs on 47 individuals:

```
 101, 102, 201, 202, 2010, 301, 302, 304, 2020, 305, 306, 307, 308,
3010, 404, 3040, 405, 406, 407, 408, 3050, 410, 411, 412, 3080, 414,
 415, 416, 4040, 505, 506, 507, 508, 4050, 509, 510, 511, 512, 4080,
 513, 514, 515, 516, 5080, 601, 5150, 602.
```

One reason for including this example is to show how the program deals with discrepancies between the list of individuals in the individuals file and those in *ibd* graphs file. Comparing with the 35-member individuals file:

- 5160 is in the individuals file, not in the output scores file; will be dropped;
- about 11 individuals are in the above list, but not the individuals file; these will be assumed unobserved, and dropped.
- another 7 individuals are in the individuals file and the above list, but the trait data indicates them as unobserved; they will be dropped.

There will remain 27 individuals who will be in the reduced ibd graph file created by `gl_lods`.

Finally we require a trait-model specification; lod scores are computed under this model. Note that the `input pedigree record` statement is used for the individuals files just as it would be for a pedigree file. The example file `ped47_D.par` provides a provides the model for a discrete trait:

```
output overwrite scores file  "./ped47_gl_D.scores"
set trait 1 data discrete
input pedigree record trait 1 integer 4
set trait 1 for tloc 11 incomplete penetrances 0.1 0.6 0.9
```

The file also specifies the output scores file '`./ped47_gl_D.scores`': the log-likelihood contributions from each IBD graph will be saved to this file.

Alternatively, the example file `ped46_Q.par` provides the models for a quantitative trait:

```
output overwrite scores file  "./ped47_gl_Q.scores"
set trait 1 data quantitative
input pedigree record trait 1 real 1
set trait 1 for tloc 11 genotype means 90.0 100.0 110.0
set trait 1 residual variance 25.0
set number of permutations 5
```

This example differs from the previous one, in that a permutation-based scheme is used to provide a locus-specific base lod score conditional on the IBD at that locus: see [GT15].

- **The simCmps example:** The program `gl_lods` will also run on data sets containing multiple components. Provided the collection of *ibd* graphs are generated by component, the `gl_lods` program will compute a lod score for each component, as does `lm_linkage`. This example is based on a pedigree file and data originally used as an `lm_linkage` gold standard. It consists of three almost identical pedigree components with very similar data. The date in `SimCmps.ind` used as a gold standard in the MORGAN `Lodscore/Gold` directory differs slightly from that in the same filename under `MORGAN_Examples/Lodscores`, and so also therefore do the output lod scores, but functionally the two are identical. The parameter file for `gl_lods` is as follows:

```
set printlevel 5

input individuals file              "./simCmps.ind"

# files for processing IBD graphs
input extra file                    "./simCmps_fgl.oscor"
output overwrite extra file         "./simCmps_fgl.reduce"

# File for outputting the log-likelihood contributions
output overwrite scores file        "./simCmps_gl.scores"

# The number of ibdgraphs provides the number of replicates in the gl_auto
# output scores file.
set maximum 100 ibdgraphs per component

# This map test tloc statement indicates at which markers lodscores are to be
# calculated.
map test tloc 11 at markers 1 3 5 6 9 10

select trait 1
set trait 1 data quantitative
input pedigree record trait 1 reals 1

set trait 1 tloc 11
set tloc 11 allele freqs 0.3 0.7

set trait 1 for tloc 11 genotype  means 485.0 500.0 515.0
set trait 1 residual  variance  10.0
set trait 1 additive  variance  10.0

# Provide externally computed base log likelihoods for each component.
set base log likelihoods -24.56078 -26.30071 -24.85028
```

The parameter statements are mostly standard specifications familiar from `lm_linkage`. There are two main differences: (1) A file of *ibd* graphs and an individuals file are specified rather than pedigree file and marker data, and (2) in order for `gl_lods` to convert its log-likelihood contributions to an estimated lod score a "base log likelihood" must be provided for each component. This should be the probability of the trait data under the model; this probability forms the denominator of the lod score. It may be computed using the `base_trait_lods` program: See Section 11.8 [Running the base_trait_lods program], page 109.

Similarly to the previous example, the `simCmps.ind` file contains the (potential) trait data on 156 individuals (3 sets of 52); in fact only a total of 55 are observed for the quantitative trait. The file `simCmp_fgl.oscor` contains 100 *ibd* graphs on the 159 individuals of the original `lm_linkage` simCmps example; this includes the 156 individuals of the current example. Note that the ordering of the IBD graphs is by component– first 100 graphs for the first component `simL_..`, then 100 for `simJ_..`, and finally 100 for `simK_...`

See [Concept Index], page 151, for: *ibd* graph, base log likelihood, individuals file.

## 11.7 Running `gl_lods` examples and sample output

- **ped47 example:** We show first some features of the `ped47` example, which exists only in the gold standards files. The `gl_lods` gold standards may be run as `make gold.6` in `MORGAN/Lodscore/Gold` or the output `ped47_gl_lods_[D/Q].gold` files may be consulted. We describe here the output file `ped47_gl_lods_D.gold`.

  The program first summarizes the input information in the usual way:

```
Trait data are discrete

Discrete data at trait locus are to be read from integer 4
   in each individual's record

Incomplete penetrances for the trait are:
       genotype  code  penetrance
       --------  ----  ----------
          (1 1)     0       0.100
   (1 2), (2 1)     1       0.600
          (2 2)     2       0.900


Number of IBD graphs requested is 1000

Number of batches for MC variance estimation is 20

=== Checking of parameter statements completed ===
```

Note that the batches for MC variance is the default as no value was provided in the parameter file: this statement is not used by `gl_lods`.

Next the information from the individuals file is checked, analogous to pedigree file checking. Since there is no pedigree, only any input genders will be scored. Also note that we specified (up to) 42 individuals, but there are only 35 in the file.

```
Opened input individuals file "./ped47.ind"

35 individuals read

42 individuals expected  (W)

Component information:

   size  females    males  unsexed    first

     35        1        1       33      302

=== Checking of individuals file completed ===
```

Then follows the usual MORGAN summary of the phenotypes of individuals:

```
Opened input individuals file "./ped47.ind"

Pedigree records contain:  names, 7 integers, 1 real number

Trait data:

Component 1:

   phenotype 2:
   302 307 406 407 408 411
   414 416 505 507 508 511
   512 513 514 516 601 602

   phenotype 1:
   306 404 410 412 415 506
   509 510 5160 5150
```

Finally it reports on the processing of the *ibd* graphs, including the result of the IBDgraph determination of equivalence classes. It also find the max number of FGL it will need: 26 in this example. It also provides '`NLoci`'; one more than the number of locations for which log-likelihoods will be computed.

```
Number of individuals in IBD graphs file is 47
1000 graphs were requested, but only 100 were given (W)

Observed individual 5160 is not in the input IBD graphs file:
  The trait data on 5160 will be ignored (W)

Number of individuals in nghd structure is 35
Have alloced gen_pen nFGL=26, NLoci=6

Opened output scores file "./ped47_gl_D.scores"

====== Computing LodScore for component 1 ======

Number of observed individuals in each IBD graph for component 1 is 27
Grouped 1292 locations into 988 equivalence classes

Log likelihoods for 100 ibd graphs at markers
  2  4  5  8  10
will be printed to the output scores file
```

Then the program produces the log-likelihood contributions at each of the 5 marker locations for each of the 100 reduced ibd graphs. Note the final warning:

```
gl_lods is not able to provide lod scores unless the user provides a base log
likelihood using the "set base log likelihood X1 ..." parameter statement (W)
```

In order to produce lod scores, rather than only log-likelihoods a base value is needed – corresponding to the marginal probability of trait data in the normal lod score computation. Since `gl_lods` does not have pedigree information, it is unable to compute this value, and it must be supplied via a parameter statement. It can be computed by running `base_trait_lods`. Note also that the log-likelihood contributions are base-e. The base log-likelihood to be input, and the final lod scores produced, are base-10.

For the quantitative trait data and model, the gold-standard output file is `ped47_gl_lods_Q.gold`. The format of this file is identical to that for the discrete trait, except that now the data and trait model are for a quantitative trait. Additionally this example uses a permutation-based approach to produce the analogue of a lod score;

see [GT15]. The gold standard uses only 5 permutations; in practice many more would be used.

```
Permutation lod scores will be computed

  Version 1 of permutation lods

Component 1 lod scores found by gl_lods:

Marker number      LodScore    StdErr

     marker-2        2.0274
     marker-4        2.7382
     marker-5        3.8031
     marker-8        1.5405
    marker-10        2.4196

  Version 2 of permutation lods

Component 1 lod scores found by gl_lods:

Marker number      LodScore    StdErr

     marker-2        1.3984
     marker-4        1.9594
     marker-5        3.3204
     marker-8        1.2037
    marker-10        2.0212
```

This permutation approach is still in process of testing: two alternative estimates are presented. Note that although this approach produces a “LodScore” the interpretation is not as for a classical lod score; see [GT15]

- **simCmps example:** This example is also a part of the `gold.6` gold standard, but may also be run in the Lodscore subdirectory of `MORGAN_Examples` using the command (Note that the `MORGAN_Examples` version is from MORGAN version 3.3: it is slightly modified in the gold standards for version 3.3.2)

```
./gl_lods simCmps_gl_lods.par > simCmps_gl.out
```

The results in the output file `simCmps_gl.out` are very similar to those for the previous example. The section summarizing the *ibd* graphs is

```
Opened input extra file "./simCmps_fgl.oscor"

Number of individuals in IBD graphs file is 159

Number of individuals in nghd structure is 156
Have alloced gen_pen nFGL=96, NLoci=7
```

```
      Opened output scores file "./simCmps_gl.scores"
```

Then for each component the log-likelihoods are computed, but this time base log-likelihoods were provided in the parameter file. The log-likelihoods are therefore converted to estimated lod scores at the 6 requested marker positions. For component 1:

```
     ====== Computing LodScore for component 1 ======

      Number of observed individuals in each IBD graph for component 1 is 19
      Grouped 1260 locations into 544 equivalence classes

      Lod scores for 100 ibd graphs at markers:
     .......

      ====== Computing LodScore for component 1 ======

      Number of observed individuals in each IBD graph for component 1 is 18
      Grouped 1260 locations into 536 equivalence classes

      Log likelihoods for 100 ibd graphs at markers
        1  3  5  6  9  10
      will be printed to the output scores file


      Component 1 lod scores found by gl_lods:

      Marker number      LodScore    StdErr

           marker-1       -6.9215
           marker-3       -7.2357
           marker-5        3.1253
           marker-6        3.4153
           marker-9       -4.8130
          marker-10       -4.7849
```

This is then repeated for components 2 and 3. There is no Monte Carlo within this non-permutation version of `gl_lods`. However, even if the three pedigree components were identical, there is Monte Carlo variation in the generation of the *ibd* graphs by the `gl_auto` program.

See [Concept Index], page 151, for: base log likelihood.

## 11.8 Running the `base_trait_lods` program

The `base_trait_lods` program provides the probability of trait data on a pedigree. (Previously, this information had to be extracted from a dummy run of the `lm_linkage` program.) The output base log-likelihoods may be input to `gl_lods`, enabling it to compute a classical normalized lod score; See Section 11.6 [Parameter files for the gl_lods program], page 101.

Typically the computation will be an exact single-locus computation by pedigree peeling. However, in the event the pedigree is too large or complex for exact computation a Monte Carlo alternative is provided. An example of each is provided in the `Lodscore/Gold` directory, and they may be run as `make gold.7`.

- Exact computation

  The input for exact computation required any a pedigree, trait data, and a specification of the trait model

  ```
  input pedigree file  "./xact.ped"
  output scores file   "./bt_lods_pp.scores"

  input pedigree record trait 1 integer 8
  set trait 1 data discrete

  select trait 1
  set trait 1 tloc 15
  set tloc 15 allele freqs 0.5 0.5

  set trait 1 for tloc 15 incomplete penetrances 0.05 0.6 0.95
  ```

  The output is self-explanatory: a base log likelihood is computed for each pedigree component. In this example there are two components, and the base log-likelihoods are also printed to the output scores file:

  ```
  1 202   -3.326957
  2 408   -1.563501
  ```

  Note the name of an index individual is also given, so that components can be correctly identified. The base log-likelihoods can now be read directly from this file into the g_lods program.

- Monte Carlo computation

  Descent is simulated on the pedigree, and the realized IBD used to compute a Monte Carlo estimate of the base log-likelihood using the `gen_pen` routine. To avoid problems with underflow on large pedigrees, some preliminary "pseudo-prior" iterations are used to provide a crude base estimate. This is then used in each term, and factored out at the end of the computation. The only difference in the parameter file is that the specified sampler seeds are used, and there are two Monte Carlo requests:

  ```
  set sampler seeds  0x53f78285 0xdfbca001 # Replace by seed file for real runs.
  # input seed file "sampler.seed"
  # output overwrite seed file "sampler.seed"
  # Monte Carlo setup and requests

  set MC realizations 3000
  set pseudo_prior iterations 150
  ```

The output is again self-explanatory. Note that, in addition to a Monte Carlo standard error, the 5th and 95th percentiles of the log-likelihood contributions are provided. once the distribution of contributions is often highly skewed, the quantiles often provide a better measure of precision than does the standard error.

```
====== Base trait LodScore for component 1 estimated using Monte Carlo ======

 Now starting 150 preliminary realizations
 Crude log-likelihood from preliminary realizations = -8.047451e+00
 Beginning 3000 realizations of Monte Carlo
 Monte Carlo realizations completed

 Number of      Monte Carlo    Monte Carlo     5th %       95th %
 realizations   lod score      StdErr          quantile    quantile

       3000         -3.3298         0.0075      -4.0353     -2.8788
```

The estimated base log-likelihoods are printed to the output scores file as before. Note that in this very small example, there is almost no difference between the exact base log-likelihoods and the Monte Carlo estimates. However, in general this will not be so; where exact computation is feasible it is to be preferred.

```
1 202   -3.329781
2 408   -1.565421
```

## 11.9 Location lod scores statements

New statements for these programs include maps for test positions, and parameters for some additional MCMC algorithms.

See [Concept Index], page 151, for: location lod scores statements, `gl_lods` statements, `lm_linkage` statements, `lm_bayes` statements.

### 11.9.1 Location lod scores computing requests

- For the '`select`' statement for your MCMC simulation, See Section 9.8.1 [Autozyg computing requests], page 74. Select all or some of the markers and '`trait 1`', and map this trait to a '`tloc 1`' (this is the trait locus to be assigned varying test positions).

See [Concept Index], page 151, for: location lod scores computing requests.

### 11.9.2 Location lod scores file identification statements

All Lodscore programs use the general MORGAN file identification statements (see Section 2.3 [File identification statements], page 11) and the Autozyg rescue file statements (see Section 9.8.2 [Autozyg file identification statements], page 75).

One additional statement is optional for `lm_bayes`:

`output Rao-Blackwellized estimates file`

If this file is specified, the set of Rao-Blackwellized lod score estimates at each trait position is written at the frequency specified in the '`compute scores`' statement.

The same standard out file specifications are available for `lm_twoqtl`. In particular:

`output extra file`
: This statement is used by `lm_twoqtl` to output batch mean estimates used in computing the estimated Monte Carlo standard error.

See [Concept Index], page 151, for: location lod scores file identification statements, Rao-Blackwellized estimates.

### 11.9.3 Location lod scores pedigree file description

Most Lodscore programs use the general MORGAN pedigree file description statements (see Section 2.9 [Pedigree file description statements], page 15). However the `gl_lods` program uses an individuals file rather than a pedigree file (see Section 2.8 [Individuals file], page 15).

The following statement used by `gl_lods` is the individuals-file analogue of the corresponding pedigree file statement:

`input individuals record names 2 [integers` *I*`] [reals` *J*`]`
: The two "names" consist of the name and component number of the individual.

In addition, there is a statement optional for `lm_linkage` and `gl_lods`:

`input pedigree record traits` *K1 K2* `...` `reals` *X1 X2* `...`
: This statement is analogous to '`input pedigree record traits` *K1 K2* `... integers` *I1 I2* `...`' (see Section 2.9 [Pedigree file description statements], page 15) when the trait is quantitative, rather than discrete. It allows the file locations of multiple traits to be defined, although only one can be selected for any analysis.

See [Concept Index], page 151, for: location lod scores pedigree file description, quantitative trait.

### 11.9.4 Location lod scores output file description

All Lodscore programs use the Autozyg output file description statements; See Section 9.8.4 [Autozyg output file description], page 76.

See [Concept Index], page 151, for: location lod scores output file description.

### 11.9.5 Location lod scores mapping model parameters

- See Section 5.4.2 [genedrop mapping model parameters], page 33, for statements specifying the genetic map for the markers.

The following statements describe the hypothesized trait locus (tloc) positions which are to be '`tested`'. That is, these are the positions at which lod scores will be computed.

`map [chromosome` *I*`] [gender (M | F)] test tloc` *L1* `all interval proportions` *X1 X2* `...`
: Interval proportions specify the proportional genetic distance between markers for the trial positions for the test trait locus. The same ratios are used between each marker pair, regardless of the inter-genetic distance (in cM).

`map [chromosome` *I*`] [gender (M | F)] test tloc` *L1* `intervals` *J1* `...` `proportions` *X1 X2* `...`

This statement specifies interval proportions, but between specific pairs of markers. Interval 1 is between markers 1 and 2, interval 2 is between markers 2 and 3, etc.

`map [chromosome` *I*`] [gender (M | F)] test tloc` *L1* `(beginning | ending | external) ([Kosambi] distances | recombination fractions)` *X1 X2* `...`

This statement specifies trial trait positions on the chromosome before the first marker and/or after the last marker.

`map test tlocs` *L1* `... no default [interval proportions| external positions]`

This pair of statements is used to eliminate computation of lod scores at default interval and/or external positions on the active chromosome.

`map [chromosome` *I*`] test tloc` *L* `at markers` *J1* `...`

This statement (new with MORGAN 3.0) is increasingly used with denser SNP marker data. If used, lod scores will be computed only at the positions of the specific markers. Note the marker indexing is by the count in the marker data file, not by selected marker.

`map [chromosome` *I*`] test tlocs` *L1 L2* `jointly at markers` *J11 J12* `...`

This statement (not yet implemented) will allow two-locus lod score programs such as `lm_twoqtl` to compute lod scores only at any specified combination of marker positions rather than, as currently, on a grid.

See [Concept Index], page 151, for: location lod scores mapping model parameters, trait test positions, map function.

### 11.9.6 Location lod scores population model parameters

- See Section 5.4.3 [genedrop population model parameters], page 34, for statements specifying the allele frequencies for the markers and trait loci, and See Section 9.8.6 [Autozyg population model parameters], page 77, for statements specifying marker names.

See [Concept Index], page 151, for: location lod scores population model parameters.

### 11.9.7 Location lod scores computational parameters

- See Section 7.4 [ibddrop statements], page 54, for setting the sampler seeds.
- See Section 9.8.7 [Autozyg computational parameters], page 77, for specifying the marker data.
- See Section 9.8.7 [Autozyg computational parameters], page 77, for specifying the trait data as genotypic, quantitative or discrete and for specifying penetrances when trait data are discrete.
- See Section 5.4.4 [genedrop computational parameters], page 35, for setting genotype means for each tloc in the case of a quantitative trait.

The following additional statements are specific to lod score computations:

`set maximum` *I* `ibdgraphs per component`

This statement is used by `gl_lods` which needs to know the upper limit of the number of ibd graphs per pedigree component provided as input.

`set base log likelihood` *X1 X2* `...`

This statement is used by the program `gl_lods`. The log of the marginal probability of the traits data on each pedigree component must be given to enable `gl_lods` to normalize its lod scores. These may be computed using the `base_trait_lods` program: See Section 11.8 [Running the base_trait_lods program], page 109.

`set number of permutations` *N*

This statement may be used by the program `gl_lods`. At each location at which a log-likelihood is computed given the realized IBD, trait-data permutation provides a normalizing null term conditional on that same IBD structure. This permutation approach is still being studied; see [GT15].

`simulate` *N* `ibd realizations`

This statement may be used by the program `base_trait_lods`, which provides a probability of trait data on a defined pedigree, under a specified trait model. If the pedigree is too complex for single-locus exact pedigree peeling, a Monte Carlo estimate may be obtained by realizing IBD on the pedigree. This statement provides the number of realizations to be used.

`set pseudo-priors` *X1 X2* `...`

This statement is optional for `lm_bayes`. The number of pseudo-priors is the number of test trait locus positions plus one. The first pseudo-prior is for the unlinked position; this should be assigned a positive value. All other pseudo-priors must be positive or zero. The set of pseudo-priors need not be normalized.

This statement is also optional for `base_trait_lods`. If the `base_trait_lods` base log-likelihood is to be estimated by Monte Carlo then this statement gives the number of realizations on which a crude estimate is made. This is then used in the main iterations to lessen the chance of over/under-flow

`set` *I* `batches MC variance estimation`

This statement is optional for `lm_linkage`, `lm_twoqtl` and `gl_lods`. These programs batch scored realizations in order to provide a Monte Carlo estimate of the standard deviation in estimating the lod score. This statement determines the batch size, and hence the number of batches. By default it is determined such that there are 20 batches.

The following additional statements are specific to tloc specification and likelihood computation for the program `lm_twoqtl`.

`set traits` *K1* `... multiple tlocs` *L1* `...`

This statement is used by the `lm_twoqtl` program to specify the tlocs *L1...* that contribute to *each* trait *K1...* A statement may be provided for each separate trait. However, the `lm_twoqtl` program expects selection of one trait with either one or two contributing tlocs.

`set trait` *K1* `for tlocs` *L1 L2* `joint genotype means X11 X12 X13 X21 X22 X23 X31 X32 X33`

This statement specifies the 9 genotypic means (3x3 matrix) for tlocs *L1* and *L2* in contributing to trait *K1*. The first index on *X* refers to the *L1* genotype and the second to *L2*.

`use [exact|MC] summation for trait`
: This statement specifies whether exact or Monte Carlo (MC) will be used by `lm_twoqtl` for computation of the trait contribution to the lod score. Exact summation can be used only on pedigrees with six or fewer founders.

`simulate` *I* `IBD realizations for trait`
: If Monte Carlo summation is to be used, this statement specifies the number of realizations of tloc inheritance realizations to be used. If Monte Carlo summation is not to be used, this statement is ignored.

`use multiplier` *I* `realizations for null`
: This statement specifies the number of time as many realizations are to be used in estimating the base-line unlinked lod-score. To obtain accurate lod-score estimates it is important this value is accurate, and it may therefore be advisable to use more realizations.

See [Concept Index], page 151, for: location lod scores computational parameters, pseudo-prior iterations.

## 11.9.8 Location lod scores MCMC parameters and options

- All the statements described in see Section 8.6 [MCMC parameter statements], page 61, for specifying the MCMC parameters are used for the location lod scores programs.

Please see that section for details regarding:

```
use (locus-by-locus sampling | sequential imputation) for setup
use I sequential imputation realizations for setup

set MC iterations I
set burn-in iterations I
sample by (scan | step)
set L-sampler probability X

check progress I MC iterations
```

As with the Autozyg programs, the number of desired MC iterations must be specified, as there is no default value.

`set MC iterations` *I*
: This statement sets the total number of '`main`' L- and M-sampler iterations. For `lm_linkage`, the total MCMC run length is the sum of the number of burn-in iterations and main iterations. For `lm_bayes`, the total MCMC run length is the sum of the number of burn-in, pseudo-prior (see below) and main iterations.

Additional statements for `lm_bayes` include the following:

`set pseudo-prior iterations` *I*
: Following burn-in, `lm_bayes` performs iterations to calculate the pseudo-priors. These pseudo-priors are used to encourage the MCMC sampler to visit test positions of low posterior probability. The default number of iterations to

compute pseudo-priors is 50% of the number of main iterations specified in the '`set MC iterations`' statement.

`set sequential imputation proposals every` *I* `iterations`
: This option applies to `lm_bayes`'s pseudo-prior and main MCMC iterations. It allows the MCMC chain to "restart" every *I*th iteration. Sequential imputation is used to propose potential restart configurations which are accepted/rejected with Metropolis-Hastings probability.

`set test position window` *I*
: This `lm_bayes` statement specifies the window size for the proposed tloc position update in the Metropolis-Hastings algorithm. *I* is the number of hypothesized trait positions on either side of the current position, with equal weight given to the 2**I* `+` 1 trait positions. The default is window size is 6.

See [Concept Index], page 151, for: location lod scores MCMC parameters and options, MC iterations, burn-in, sequential imputation proposals.

# 12 Population-based inference of IBD

See [Concept Index], page 151, for: population-based IBD inference.

## 12.1 Introduction to ibd_create and ibd_haplo

See [References], page 149, for details of the cited papers.

The program `ibd_create` is a suite of seven subprograms that together provide a set of tools for the creation of haplotypes and *ibd* specifications, including *ibd* graphs (See Section 4.6 [The ibd_class utility], page 26). These programs produce realistic simulated data for use in testing analysis programs such as `ibd_haplo`. Similarly to the `ibd_class` utility (See Section 4.6 [The ibd_class utility], page 26), `ibd_create` calls each of its seven subprograms through a command line option. This option also determines which MORGAN parameter statements will be recognized and how they will be interpreted.

All the examples for `ibd_create` and `ibd_haplo` are based on the current gold standards. The marker data for these gold standards have been updated to the publicly available European samples of the 1000 Genomes project. These provide 758 phased and imputed haplotypes based on the GBR, FIN, IBS, CEU and TSI subpopulation samples. The phased haplotypes were downloaded from the Browning website, http://bochet.gcc.biostat.washington.edu/beagle/1000_Genomes.phase1_release_v3/ while allele frequency and map position information were obtained from the original 1000 genomes vcf files: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20110521/ (See http://mathgen.stats.ox.ac.uk/impute/README_1000G_phase1integrated_v3.txt for additional information.) A BEAGLE DAG model was constructed for these haplotypes by running BEAGLE with a scale parameter 1.0. All files with the partial name "eur_recode_s1" derive from this data set and DAG model.

For greater clarity we here divide the programs into three subgroups. The first two subprograms are:

- **beaglesim**: A program written by Chris Glazner to simulate independent realizations of haplotypes out of a BEAGLE DAG [Browning06]. The input to BEAGLE is a set of real SNP haplotypes from a population sample. The goal of `beaglesim` is to realize haplotypes on this set of real SNPs, at known real SNP locations, with the sample SNP allele frequencies, and with the LD structure as fit by BEAGLE to the original sample.
  - `beaglesim` also allows an “LD relaxation parameter” to allow haplotypes to be generated from the same DAG but at varying LD levels.
  - However, for purposes of realistic simulation studies, it is recommended that BEAGLE be run with the low scale parameter 1.0 (*not the value 4.0 which is the default for the BEAGLE 3.3.1 option that produces the DAG*), and that `beaglesim` then be run without additional LD tuning (i.e. the LD relaxation probability parameter set equal to 0).
  - `beaglesim` can take and produce data either as character (A,C,G,T) or numeric (1=reference allele, 2=alternate allele). For downstream use in other MORGAN programs, it will be found more convenient to have converted to numeric allele labels.

- Raw data haplotypes are typically coded with a row for each SNP, and a column for each haplotype. `beaglesim` generates realizations by haplotype, but can output these haplotypes either as rows or as columns or in MORGAN marker data genotypic format (See '`marker data`' in [Concept Index], page 151). Care must be taken in ensuring the correct row/column specification for downstream use in other MORGAN programs, or other software.

For additional information about `beaglesim` see [BGZT12].

- **beagledag**: This is a prototype program to parse and compute local haplotype probabilities given a BEAGLE DAG model. In its current form, it provides a list of haplotypes and their DAG probabilities for all feasible local haplotypes between two specified markers. Some of the routines of this program will be useful in developing fast methods to compute the DAG probabilities of specific local haplotypes. These probabilities could be used to weight realized IBD segments inferred using `gl_auto` or `ibd_haplo`. This would provide an adjustment for LD when this inferred *ibd* is used in `gl_lods` to produce Monte Carlo lod score estimates (See Section 11.6 [Parameter files for the gl_lods program], page 101). (See Section 4.6 [The ibd_class utility], page 26, for more information on *ibd* graphs).

The next subprogram of `ibd_create` is:

- **simpop_fgl**: This is a version of a program originally written by Chris Glazner, and modified by Fiona Grimson. It simulates both a population pedigree, and the crossover process on a chromosome. With MORGAN version 3.3.2 the program is further modified, so that the cross-over process is in terms of (micro)-centiMorgans rather than base pairs. The program produces an *ibd* graph of current individuals in terms of a compact specification of the FGL segments of their maternal and paternal gametes (See Section 4.6 [The ibd_class utility], page 26, for more information on *ibd* graphs).
  - A population of diploid size N is simulated, with N/2 males and N/2 females in each generation. For each of N times, a random male and random female is chosen, and the couple produces a son and a daughter.
  - A modification due to Fiona Grimson reduces the numbers of half-sibs and multiple matings. A parameter m defines the remating probability. If a selected individual has been selected k times previously as a parent, he/she is accepted again with probability m to the power k. With m=0.3, this gives an effective population size close to the census size – the greater variance of offspring number caused by families of size 2, compensated by the less-than-Poisson variation in the number of matings of a parent.
  - In any meiosis, crossovers between the two parental gametes are generated at rate 1 per Morgan (100 million micro-centiMorgans) across the chromosome, and these crossover events are used in generating the FGL segments of the offspring gamete.
- **simped_fgl:** This subprogram is no longer in `ibd_create` It is replaced in MORGAN V3.4 by a new version of `ibddrop` See Section 7.1 [Introduction to ibddrop], page 46. The new `ibddrop` both simulates descent at a finite number of linked markers, but also can now simulate the recombination breakpoints across the chromosome.

The final three subprograms of `ibd_create` are

- **fgl2ibd**: An initial version of this program for sets of four gametes was first written in C by Fiona Grimson. This was later generalized focusing initially on sets of 6 gametes, but later generalized to (in principle) any number. The program has three inputs:
  1. The key input is an *ibd* graph for a set of gametes specified in compressed FGL form; with MORGAN 3.3.2 the recombination breakpoints of the graph are in micro-centiMorgans.
  2. Also input is a specified set of SNP marker locations. These may be input either
     - on the cM scale via standard MORGAN map statements, or
     - on the base-pair (bp) scale.

     In the latter case, a parameter statement is provided to covert from the bp scale to the cM of other MORGAN programs.
  3. The MORGAN program `set proband gametes` statements are used to define the gametes among which `ibd` is scored.

  The output is `ibd` states for the specified gametes, at the specified marker locations: this output is in a variety of formats specified more fully below.
- **fgl2haplo**: This program assigns haplotypes to individuals on the basis of their location-specific IBD as specified by the FGLs of the *ibd* graph. The haplotypes are given at specified SNP markers. The data generated are typically used in testing analysis methods. The program has been updated, so that repeated runs use different random assignments of founder haplotypes to the FGL.
  - The inputs to the program include an *ibd* graph in compact FGL form, a map of locations of selected markers at which haplotypes are to be scored.
  - Also input is a set of haplotypes to be assigned to these FGLs. The haplotypes are typically either from a public data base or generated by `beaglesim` from a BEAGLE DAG that was fit to such haplotypes. Each FGL is assigned a unique haplotype. To permit multiple realizations on a single IBD structure, the set of haplotypes is randomly permuted before the assignment is made.
  - The output of the program is the haplotypes of the individuals of the *ibd* graph, constructed by assigning segments of the input haplotypes to the individuals in accordance with the defined FGL structure.
  - The output haplotypes may be as rows or as columns, or in MORGAN marker data genotypic format.
- **fgl2dgl**: This program takes as input the *ibd* graph in the compact FGL form with distances in micro-centiMorgans, as produced by `simpop_fgl`. Its output is an *ibd* graph in the format produced by the `gl_auto` program, which can therefore be input into `gl_lods`; See Section 9.1 [Introduction to lm_auto gl_auto and lm_pval], page 63.

**The program** `ibd_haplo` computes conditional probabilities of gene IBD (identity by descent) states, given data for marker loci for specified sets of proband gametes (i.e. *scoresets*). The proband gametes in each scoreset are specified gametes (maternal or paternal) of specified individuals. These are input as for the `lm_auto` program: See Section 9.2 [Sample lm_auto parameter file], page 64.

The program has been generalized to allow for sets of up to ten gametes, although computational limitations suggest that considering more than seven gametes jointly is impractical.

Internally, IBD states are the gametic states (that is, 15 states for 4 gametes), and states are ordered lexicographically, although the "traditional" Jacquard ordering may be requested for sets of four gametes.

The marker data are read in using a standard MORGAN marker data file; See Section 9.2 [Sample lm_auto parameter file], page 64. Each individual named in the marker data must have a unique name. There may be missing data, but each single-locus genotype of an individual must be either present or absent(`" 0 0"`); presence of a single allele cannot be specified. The marker data are read in as genotypes, but they may be analyzed as an ordered pair of alleles (i.e. phased), and must be so if only a single gamete of any individual is specified in the scoreset. (??)

The program uses a HMM model for the latent IBD states. There are two options for the transition matrix of the HMM latent IBD state; the one applicable to any number of gametes is the '2011' matrix developed by Chaozhi Zheng [BGZT12]. Given the latent state, the locus-specific genotype probabilities are based on the premise that IBD DNA should be of the same allelic type, and that non-IBD DNA is of independent allelic types, although allowance is made for typing error to eliminate zero emission probabilities. The transition matrices are also modified to eliminate zero transition probabilities. A forward-backward HMM computation provides the probabilities for each IBD state at each locus for each set of proband gametes.

For the case when the scoreset consists of both maternal and paternal gametes of a set of (normally two) individuals, additional options are available. For two individuals there are 15 IBD states, although only 9 are distinguishable from unphased genotypes. In this case there are two transition matrices available; the earlier '2009' matrix of [Tho08b] is also an option. The input genotypes may be analyzed as an ordered or unordered pair of alleles (i.e. phased or unphased). There is also an option for partial phasing, in which segments of chromosome (sets of contiguous markers) are specified as phased. If the data are analyzed as unphased or partially phased, the output state probabilities are of the genotypically distinguishable state classes (i.e. 9 states instead of 15). Finally, although internally the program still works with a lexicographic ordering of states, for four gametes output may be in terms of the more conventional ordering of the Jaquard states.

The methods and study results of this approach are provided in [Tho08b] and [BGZT12]. Note that the data files and software released for [BGZT12] are for an earlier version of `ibd_haplo`. The version described here includes improvements both in user interface, and also in the way the IBD transitions are implemented. This version was first released for MORGAN V3.1.1 with more minor improvements for MORGAN 3.2. Specifically, for MORGAN 3.2 there have been modifications in the computation of locus-to-locus transition probabilities, providing for better approximation to the underlying continuous process modeled in [BGZT12].

See [Concept Index], page 151, for: `ibd_create` introduction, `ibd_haplo` introduction, marker data, proband gametes, *ibd* graph

## 12.2 Sample parameter files for ibd_create; `beaglesim` and `beagledag`

- Here is a parameter file for `beaglesim`:

```
    # Include everything in the output file.
```

```
set printlevel 5

# Provide a file name for the beaglesim input DAG file.
input extra file              "./eur_recode_s1.bgl.dag.gz"

# Provide a file name for the beaglesim haplotypes file.
output overwrite scores file  "./eur_recode_s1.haplotypes"

# The sampler seeds are going to be 53 and 5353 (ie '0x35' '0x14e9').
set sampler seeds 53 5353     # Set the sampler seeds.

# The following Morgan parameter statements are needed by beaglesim.

output 758 haplotypes as rows
set LD relaxation probability 0.05
```

These statements are mostly self-explanatory: See Section 12.10.1 [beaglesim and beagledag parameter statements], page 130, for more details.

- Here is a parameter file for `beagledag`:

```
# Include everything in the output file.
set printlevel 5

# Provide a file name for the beagledag input DAG file.
input extra file              "./eur_recode_s1.bgl.dag.gz"

# Provide a file name for the beagledag reduced DAG file.
output overwrite scores file  "./eur_recode_s1.reduced"
```

Note that `beagledag` is still a prototype program. This particular version is hard-wired to compute local haplotype frequencies for a specific subset of consecutive markers from this DAG. The subroutine structures are more general and a more flexible example of the calling program will be released in the future.

See [Concept Index], page 151, for: sample parameter file for `beaglesim` and `beagledag`, using the BEAGLE DAG.

## 12.3 Running ibd_create examples and sample output; beaglesim and beagledag

The `ibd_create` subprograms are called by invoking the program with a flag specific to the subprogram. Examples files are included in the `Haplo` subdirectory of `MORGAN_Examples`. Here are the relevant run commands

*`./ibd_create -s beaglesim.par`*

Running `ibd_create` with the `-s` options runs `beaglesim`. The example runs on the BEAGLE DAG [Browning06] that was fit to 758 European haplotypes of the 1000 Genomes Data. Haplotypes are generated from the DAG model. It is recommended that both the output haplotypes and the unzipped version of the DAG file are removed after the program is run: these are large files.

*./ibd_create -d beagledag.par*
: Running `ibd_create` with the `-d` options runs `beagledag`. This program computes local haplotype frequencies for a specific set of markers in the DAG file. It is still very much a prototype program; it may be generalized in the future. It is recommended that both the output reduced DAG file end the unzipped version of the DAG file are removed after the program is run: these are large files.

See [Concept Index], page 151, for; running `beaglesim` and `beagledag`.

## 12.4 Sample parameter files for ibd_create; simpop_fgl

The formats for `simpop_fgl` have been modified for MORGAN version 3.3.2 and subsequent. This subprogram now works in centiMorgans rather than base pairs. The Gold standard parameter file is identical to the the one in the `MORGAN_Examples/Haplo` subdirectory.

- Here is a parameter file for `simpop_fgl`:

```
set printlevel 3

# Provide a file name for the simpop_fgl results file.
output overwrite scores file  "./simpop_fgl.ibdgraphs"

set sampler seeds 11 7654                      # Set the sampler seeds.

# The following Morgan parameter statements are being used to pass the integers
# and double precision reals needed as arguments for the simpop_fgl subprogram.

set 19 females per generation
set 30 offspring generations
output final 2 generations

set chromosome length 120.0 centiMorgans
set 0.9 remating probability weight
```

These statements are mostly self-explanatory: See Section 12.10.2 [simpop_fgl parameter statements], page 131, for more details, including the control of individuals as parents through the `remating probability`.

See [Concept Index], page 151, for; sample parameter file for `simpop_fgl`, simulation of a population, simulation of descent in a pedigree.

## 12.5 Running ibd_create examples and sample output; simpop_fgl

The examples here are the ones in the `IBD_Haplo/Gold` directory. They may be run as the gold standards `simpop_fgl_gold` which is a part of `gold.4` in this directory. Alternatively they may be run directly in the Gold directory as follows:

- ../ibd_create -p simpop_fgl.par

  Running `ibd_create` with the `-p` options runs `simpop_fgl`.

In each case, a file containing the *ibd* graph of the requested individuals is produced. Note that these *ibd* graphs have a slightly different format from those used in the `gl_auto` and `gl_lods` programs. They are indexed in micro-centiMorgans, and contain additional information about the simulated population pedigree.

The `fgl2dgl` subprogram provides a translation between these two forms of *ibd* graph.

See [Concept Index], page 151, for; running `simpop_fgl`, *ibd* graph.

## 12.6 Sample parameter files for ibd_create; fgl2ibd and fgl2haplo and fgl2dgl

See [References], page 149, for details of the cited papers.

There are two basic alternative requests for the `fgl2ibd` subprogram of `ibd_create`. One requests scoring of *ibd* among all pairs of individuals; the other, used in this example, requests *ibd* scoring for specified sets of proband gametes. This example requests sets size 2, 3, 4, 5, 6, 7 and 10; the scoreset names indicate this but the names are arbitrary.

In this example, the marker data file provides marker locations in centiMorgans: thus any scaling factor to convert from base pairs to a genetic map is unnecessary and (if included) has no effect, although the output will report the scaling factor..

Here is the `fgl2ibd_varying.par` parameter file for this example:

```
# Provide the simulated fgl (simpop_fgl's output) file name.
input gamete data file        "./simpop_fgl.ibdgraphs"

#  Each FGL in the data set is assigned a unique haplotype.
#  The "sampler" seeds are used to randomly permute the haplotypes before
#     assignment, to permit multiple realizations on a single IBD data set.

set sampler seeds 0x00003039 0x00000431  # Replace by a seed file for real runs.

# Provide a marker data file name of a file containing the marker map.
input marker data file        "./eur_recode_s1_map.markers"

# Provide a file name for the fgl2ibd results file.
output overwrite scores file  "./fgl2ibd_varying.statelabels"

# Provide a file name for the fgl2ibd score set identifiers file.
output overwrite extra file   "./fgl2ibd_varying.ids"
```

```
# The following Morgan parameter statements are being used to specify which
# proband gametes to include in each analysis run by the fgl2ibd subprogram.

set scoreset 21 proband gametes 1148 0  1161 0

set scoreset 31 proband gametes 1148 0  1158 1  1162 0

set scoreset 41 proband gametes 1158 0  1158 1  1168 0  1168 1

set scoreset 51 proband gametes 1148 0  1151 0  1158 0  1161 0  1168 0

set scoreset 61 proband gametes
1158 0  1158 1  1161 0  1161 1  1168 0  1168 1

set scoreset 71 proband gametes
1148 1  1151 1  1158 1  1161 1  1168 1  1171 1  1178 1

set scoreset 101 proband gametes
1148 0  1148 1  1151 0  1151 1  1158 0  1158 1  1161 0  1161 1  1168 0  1168 1


# The following Morgan parameter statement specifies the marker locations
# at which ibd will be scored.

select markers  10    20    30    40    50    60    70    80    90   100
               110   120   130   140   150   160   170   180   190   200
               210   220   230   240   250   260   270   280   290   300
               310   320   330   340   350   360   370   380   390   400
               410   420   430   440   450   460   470   480   490   500
.........
              4410  4420  4430  4440  4450  4460  4470  4480  4490  4500
              4510  4520  4530  4540
```

For the `fgl2haplo` program there are several options regarding the format of the input haplotypes. Here we give one example where the input haplotypes are specified as columns. The marker map is in centiMorgans so any scaling factor declared is irrelevant. However tha input value will be reported in output (or 1.0 will be reported, if the statement is not included)..

```
# Provide the ibd graph input (simpop_fgl's output) file name.
input gamete data file        "./simpop_fgl.ibdgraphs"

# Provide the output haplotype labels and output with spaces options.
output haplotype labels
output with spaces

# The following Morgan parameter statement specified the markers
# over which haplotypes will be generated.  (Typically one will generate
# complete haplotypes.)

select all markers

# The following Morgan parameter statement is irrelevant unless the input
#  marker map is specified in base pairs.  For a centiMorgan map there is
#  no scaling.   However the output will report the input value, or
#  1.0 if the statement is not included.

set 1.2 million base pairs per centiMorgan

# Provide a marker data file name of a file containing the marker map.
input marker data file        "./eur_recode_s1_map.markers"

# Provide the founder haplotypes file name.
input extra file              "./eur_recode_s1_hapcols"

# Tell fgl2haplo how to interpret the haplotype data read in from the founder
# haplotypes file.

input 758 haplotypes as columns of 4547 snps

# Provide a file name for the fgl2haplo results file.
output overwrite scores file  "./fgl2haplo_haps_as_cols.results"
```

The `fgl2dgl` subprogram converts the ibdgraphs file from simpop to the form used gy the `gl_lods` program; See Section 11.6 [Parameter files for the gl_lods program], page 101. The output file scores switchpoints at marker locations. In this example the marker locations are specified in base pairs, and a non-standard scaling is used in the conversion to illustrate the use of this statement. This is the `fgl2dgl_bp_0.75.par` file:

```
# Provide the simulated fgls (simpop_fgl's output) file name.
input gamete data file        "./simpop_fgl.ibdgraphs"

# Provide a marker data file name of a file containing the marker map.
input marker data file        "./fgl2dgl_bp_map.markers"

# Provide a file name for the fgl2dgl results file.
output overwrite scores file  "./fgl2dgl_bp_0.75.ibdgraphs"

# The following Morgan parameter statement is required in order to satisfy
# the proc_auto subroutine.

select markers   120   130   290   450   525   770   979   980  1091  1290
                1492  1597  1726  1900  2116  2400  2810  3055  3250  3400
```

```
# The following Morgan parameter statement is now optional.  If the user needs
# to specify something other than 1.0 million base pairs per centiMorgan, it
# can be used to pass a double precision real number as a scale factor to be
# used by the fgl2dgl subprogram.

set 0.75 million base pairs per centiMorgan
```

See [Concept Index], page 151, for; sample parameter files for `fgl2ibd`, `fgl2haplo` and `fgl2dgl`.

## 12.7 Running ibd_create examples and sample output; fgl2ibd and fgl2haplo and fgl2dgl

These two programs have a variety of input formats and output options. Here we include only one example of each program. For additional information see the README_userdoc file in `MORGAN/IBD_Haplo` of Section 12.10.3 [fgl2ibd and fgl2haplo and fgl2dgl parameter statements], page 132. All three programs use as input a file of *ibd* graphs in the format produced by `simpop_fgl`.

*`./ibd_create -i fgl2ibd_varying.par`*

Running `ibd_create` with the `-i` flag runs `fgl2ibd`. The program `fgl2ibd` simply scores the `ibd` in that file at specified markers for specified individuals. Note that you should retain (or re-create) the `simpop_fgl` output file, `simpop_fgl.ibdgraphs` which is used as an input in `fgl2ibd`. The program produces two output files. The first specifies only the scored proband gametes, while the second gives the state specification at each marker for each proband gamete set. This separation simplifies downstream analysis, particularly if using R. Although for this example the output files are not large, it is good practice to remember to remove the output files after running the program.

*`./ibd_create -f fgl2haplo_hapcols.par`*

Running `ibd_create` with the `-f` flag runs `fgl2ibd`. The program `fgl2haplo` uses the supplied *ibd* graphs (for example those produced by `simpop_fgl` and supplied haplotypes to generate genetic marker data for specified individuals, in accordance with their *ibd* across the chromosome. Note that the "haps_as_cols" refers to the input format of haplotypes, not the output. Output is either as haplotypes or genotypes in rows. Remember to remove any large output files after running the program.

*`./ibd_create -g fgl2dgl_bp_0.75.par`*

Running `ibd_create` with the `-g` flag runs `fgl2dgl`. This program takes the supplied *ibd* graphs in the format produced by `simpop_fgl`, and produces an *ibd* graph (or *dgl* graph) in the format produced by `gl_auto` and used in programs such as `gl_lods`. The input has switch-points in micro-centiMorgans. The output form provides switch-points at selected markers, so that, if the marker map is in base-pairs rather than centiMorgans, the program makes the appropriate conversion.

See [Concept Index], page 151, for; running `fgl2ibd`, `fgl2haplo` and `fgl2dgl`, phased and unphased genotypes.

## 12.8 Sample parameter files for ibd_haplo

Three sample parameter files for `ibd_haplo` can be found in the directory `MORGAN_Examples/Haplo`. All three examples are based on examples in the Gold standards for the program. The examples are analogous to the three examples previously used, except that the data have been changed to use (simulated) individuals and haplotypes from the 1000 Genomes data.

The first two examples (`phased_2011.par` and `unphased_2011.par`) use the same data, and score the same sets of 4 gametes, consisting of the maternal and paternal gametes in five pairs of individuals. The examples differ only in whether the data are treated as phased haplotypes or unphased genotypes.

Here is the `unphased_2011.par` parameter file:

```
set printlevel 3     # See comment below

input marker data file       "./sim76indivs.markers"
output overwrite scores file "./unphased_2011.qibd"
output overwrite extra file  "./unphased_2011.ids"

# The following five Morgan parameter statements specify the
#   computational set-up for the program

select 2011 state transition matrix
select unphased data

set population kinship             0.05
set kinship change rate            0.05
set transition matrix null fraction 0.05

set genotyping error rate          0.01
output four-gamete state order jacquard

# The following Morgan parameter statements are being used to specify which
# proband gametes to include in each analysis run by ibd_haplo.

set scoreset   1 proband gametes  1107 0 1107 1 1119 0 1119 1
set scoreset   2 proband gametes  1111 0 1111 1 1115 0 1115 1
set scoreset   3 proband gametes  1123 0 1123 1 1127 0 1127 1
set scoreset   4 proband gametes  1131 0 1131 1 1135 0 1135 1
set scoreset   5 proband gametes  1169 0 1169 1 1170 0 1170 1

# The program computes ibd at, and uses only,  a subset of the markers:
#  in fact every tenth marker in this example
```

```
select markers  10    20    30    40    50    60    70    80    90   100
               110   120   130   140   150   160   170   180   190   200
               210   220   230   240   250   260   270   280   290   300
.... lines omitted here
              4310  4320  4330  4340  4350  4360  4370  4380  4390  4400
              4410  4420  4430  4440  4450  4460  4470  4480  4490  4500
              4510  4520  4530  4540
```

Since ibd_haplo typically runs with a very large number of markers is is advisable to suppress printing of marker map and allele frequencies using the 'printlevel' setting. The file specifications, and marker data, are as for previous programs such as lm_auto: See Section 9.2 [Sample lm_auto parameter file], page 64. Note that there are two output files.

The marker data file sim76indivs.markers contains the positions and allele frequencies of 4547 markers and marker genotypes of 76 individuals. (These data were created using simpop_fgl and fgl2haplo, starting from the BEAGLE DAG fit to the European chromosomes of the 1000 Genomes data. In this example a subset of the markers is selected– see the end of the file.

The second group of statements relate to the ibd_haplo implementation. The '2011' transition matrix is to be used; this is the one described in [BGZT12] and is recommended. The earlier '2009' option of [Tho08b] is retained for backwards compatibility. The data are to be analyzed as unphased genotypes, and there are four numerical parameters of the HMM model. Most importantly these include the 'population kinship', which is the mean *a priori* level of pairwise IBD between any pair of gametes.

For compatibility with previous output formats, the parameter file requests ordering of the states in all output information in the "genetic" (or Jacquard) order, rather than lexicographic order. Note that this option is only available for sets of four gametes. Here the four gametes are the two of each of two individuals, and data are analyzed as unphased, so there will be nine states in the output IBD probabilities. (The ordering of the states is given at the end of Section 12.9 [Running ibd_haplo examples and sample output], page 129.)

The next set of statements specifies four sets of four gametes among which IBD is to be scored. Since the data are to be analyzed as unphased it is required that each set contains both the maternal and paternal gametes of individuals. In general, any gametes of individuals in the marker data file may be specified.

The second example parameter file phased_2011.par differs only in the names of the output file and in the statement 'select phased data', which specifies that the data should be treated as phased haplotypes. In this case it is not necessary that a scoreset consists of both gametes of individuals.

The third example ten_ss.par shows the flexibility of scoresets. This example differs in that only a smaller subset of markers is used:

```
select markers 3010  3020  3030  3040  3050  3060  3070  3080  3090  3100
               3110  3120  3130  3140  3150  3160  3170  3180  3190  3200
               3210  3220  3230  3240  3250  3260  3270  3280  3290  3300
```

Additionally, the scoresets are quite varied:

```
set scoreset 61  proband gametes  1174 0 1174 1 1176 0 1176 1 1163 0 1163 1
set scoreset 41  proband gametes  1163 0 1163 1 1169 0 1169 1
set scoreset 43  proband gametes  1165 0 1165 1 1167 0 1167 1
set scoreset 51  proband gametes  1163 0 1163 1 1169 0 1169 1 1165 0
set scoreset 32  proband gametes  1163 0 1163 1 1169 0
set scoreset 21  proband gametes  1176 0 1176 1
set scoreset 42  proband gametes  1174 0 1174 1 1176 0 1176 1
set scoreset 52  proband gametes  1163 1 1165 1 1167 1 1169 1 1170 1
set scoreset 44  proband gametes  1170 1 1170 0 1172 0 1172 1
set scoreset 31  proband gametes  1167 0 1169 1 1169 0
```

The scoresets may have arbitrary numerical indicators, and range in size from 2 to 6 gametes. The program will reorder them according to size.

Note that the Jacquard ordering is requested for the four-gamete scoresets. There are three such scoresets; these will use Jaquard ordering, while for the remainder the ordering will be lexicographic. This again shows the flexibility of the program, but mixing orderings is likely to be confusing in real analyses.

## 12.9 Running ibd_haplo examples and sample output

Run the examples in the `Haplo` subdirectory of the `MORGAN-examples` directory with the following command

```
./ibd_haplo  unphased_2011.par > unphased_2011.out
or
./ibd_haplo  phased_2011.par > phased_2011.out
or
./ibd_haplo  ten_ss.par > ten_ss.out
```

Each example produces two output files in addition to the standard output. The standard output gives little information when '`printlevel 3`' is used. The proband gamete sets are specified, and the program reports as it analyzes each set. In between, the program does give the prior probability of IBD states for each size of scoreset requested, and the transition matrix at a distance of 1 centiMorgan – these are mainly for checking purposes.

The two main output files are the '`qibd`' file and the '`ids`' file. For the first example, these have been named as `unphased_2011.qibd` and `unphased_2011.ids` with analogous names for the other examples. (Of course, any names for these files can be specified in the parameter file.) Each file contains only numeric data, so that it can be read easily into R or other programs for further analysis. The '`qibd`' file is the key output of probabilities of IBD states in each scoreset at each marker, computed conditional on marker data. The '`ids`' file gives the scoresets.

For this example the '`ids`' output file is

```
1  4  9  1107 0  1107 1  1119 0  1119 1
2  4  9  1111 0  1111 1  1115 0  1115 1
3  4  9  1123 0  1123 1  1127 0  1127 1
4  4  9  1131 0  1131 1  1135 0  1135 1
5  4  9  1169 0  1169 1  1170 0  1170 1
```

That is, there are five scoresets numbered 1 to 5. Each consists of 4 gametes. These gametes are specified in the usual format: '`0`' for a maternal gamete and '`1`' for a paternal, of the individual whose name ID is given. Since the data are analyzed as unphased, there are only 9 IBD states for each scoreset.

The number of lines in the output '`qibd`' file is the number of scoresets times the number of markers used: 2270 for the first two examples here. For the '`unphased`' case, each line consists two integers, followed by 10 real numbers. The first line starts

```
    1    10    0.618463   0.0000  0.0000  0.0028  0.0003  0.0028  0.0003  0.8416 ....
```

while line 2037 starts

```
    5  2210   58.999373   0.1462  0.0009  0.2902  0.0011  0.1479  0.0007  0.0095 ....
```

The first item indicates the scoreset, the second the marker number, and the third the centiMorgan (or Mbp) position of the marker. The remaining 9 numbers are the probabilities of the 9 IBD states. In the first line, most of the probability (0.8416) is in state-7, which is the state '`1212+1221`'. That is the two individual's are likely to share both gametes IBD at this first locus, but in this state there is no IBD between the gametes within individuals. In the second example, at marker 2210 in scoreset 5, these is probability 0.1462 that all four gametes of the two individuals are IBD, and even higher probability (0.2902) that the two gametes of the first individual are IBD, and shared IBD with one of the two gametes of the second individual.

For sets of 4 gametes, we use the traditional ordering of the 15 IBD states or 9 reduced genotypic states:

```
The order of the 15 states is 1111, 1122, 1112, 1121, 1123, 1211, 1222,
1233, 1212, 1221, 1213, 1231, 1223, 1232, 1234.

For the nine reduced states, the order is the same,
but genotypically equivalent ones are
combined:  1111, 1122, 1112+1121, 1123, 1211+1222,
1233, 1212+1221, 1213+1231+1223+1232, 1234.
```

For more general gamete sets, the ordering is lexicographic.

For more on the specification of IBD states see Section 9.2 [Sample lm_auto parameter file], page 64.

## 12.10 Population-based IBD inference parameter statements

See [Concept Index], page 151, for population-based IBD inference parameter statements

### 12.10.1 beaglesim and beagledag parameter statements

The following statements are specific to the `beaglesim` and `beagledag` subprograms of `ibd_create`, or have a particular role in these subprograms.

`set LD relaxation probability` *`X`*

`beaglesim` can simulate haplotypes from the same base DAG but at varying (lesser) levels of LD. At each DAG node, with a probability equal to the parameter, the program selects a random node at the next level, breaking LD in the generation of this particular haplotype. The default value 0.0 is generally recommended, except where these varying LD levels are the target of analysis.

`output haplotypes as (rows | columns)`

`beaglesim` takes the DAG model produced by the BEAGLE software, and simulates haplotypes from the model. These haplotypes are generated one-by-one as “rows”, but may be output either as rows or a columns for easier use in downstream programs.

`output` *I* `genotypes`
: This is an alternate output option for `beaglesim`. Note that either an `output haplotypes ,,,` or `output ... genotypes` is required.

  `beaglesim` can output its haplotypes in MORGAN marker data file format, so that they can be more easily used in MORGAN downstream analyses. The output produced by `beaglesim` in this case consists of a `set markers ... data ...` statement with the gametes ordered as defined for this parameter statement.

`output with spaces`
: This statement can be used to insert spaces between the alleles in the output file of haplotypes produced by `beaglesim`. The file is twice as large, but it may be easier for input to downstream analysis programs. The default is absence of spaces.

`output haplotype labels`
: This statement implies the "output with spaces" option for haplotype output. If haplotype labels are requested, and if haplotypes are to be output as rows, then each row will include as the first item the ID number of the individual to whom the haplotype belongs. If labels are requested, and if haplotypes are output as columns, then the first line of the output will contain the ID numbers of the individuals to whom the corresponding haplotype column belongs.

### 12.10.2 simpop_fgl parameter statements

The following statements are specific to the `simpop_fgl` subprogram of `ibd_create`, or have a particular role in these subprograms.

`set chromosome length` *R* `centiMorgans`
: This statement is required by `simpop_fgl`.

  It provides the total length of the chromosome in which these programs simulate recombination breakpoints at each meiosis.

`set` *I* `offspring generations`
: This statement is required by `simpop_fgl`.

  This sets the number of additional generations after the founder generation that `simpop_fgl` will generate.

`set` *I* `females per generation`
: This statement is required by `simpop_fgl`.

  `simpop_fgl` simulates generation of constant size, with equal numbers of males and females in each generation. This parameter specifies the number of females in each generation.

`set` *X* `remating probability weight`
: This statement is optional for `simpop_fgl`.

  `simpop_fgl` simulates each generation by successively selecting a random male and a random female and generating a male and a female offspring. If a sampled individual has been a member of $m$ previous matings, then the selected individual is accepted with probability *X^m*. The value *X = 1/3* generates reasonable human pedigrees, and gives an effective population size about equal to the census size. (Random mating is achieved by the default parameter value 1.0, but is not recommended.)

`output final` *I* `generation`
: This statement is required by `simpop_fgl`.

  `simpop_fgl` outputs only the last generations that it simulates. This parameter specifies the number of these final generations that will output.

## 12.10.3 fgl2ibd and fgl2haplo and fgl2dgl parameter statements

The following statements are specific to the `fgl2ibd` and `fgl2haplo` subprograms of `ibd_create`, or have a particular role in these subprograms.

`map [chromosome` *I*`] marker positions base pairs` *X1 X2*`...`
: Although genetic-map (centiMorgan) locations are to be preferred where available, the `fgl2haplo`, `fgl2ibd`, and `fgl2ibd` subprograms can alternatively be provided with marker locations in base pairs.

`set` *R* `million base pairs per centiMorgan`
: The `fgl2haplo`, `fgl2ibd`, and `fgl2ibd` subprograms can provide marker locations in base pairs. In this case, the scaling factor *R* provided by this statement is used to convert these locations to the centiMorgan scale used by `simpop_fgl`. If no scaling is provided, the program equates 1 million base pairs with 1 centiMorgan.

`input gamete data file` *S*
: This statement is used by both `fgl2ibd` and `fgl2haplo` to specify the input file from which the program should read the fgl-segment compact-format chromosomes that it uses. Currently it is assumed that the chromosomes are in the *ibd* graph format generated by `simpop_fgl`. This is a slightly different format from the *ibd* graphs produced from analysis of marker data on a defined pedigree that are produced by the `Autozyg` program `gl_auto`. (See Section 4.6 [The ibd_class utility], page 26).

`input` *I* `haplotypes as (columns | rows) of` *I2* `SNPs`
: `fgl2haplo` requires a set of marker haplotypes which it will apply to the fgl-based *ibd* graph. These may be input either as rows or as columns, but the number of SNPs and haplotypes should be specified.

`input genotypes as rows`
: Alternatively, the input haplotypes for `fgl2haplo` may be input in standard MORGAN marker genotype format. In this case each row of marker genotypes will be interpreted as pair of phased haplotypes. (See Section 8.4 [Single and multiple meiosis LM-samplers], page 58, and `‘‘phased and unphased marker haplotypes’’` in [Concept Index], page 151)

`output all genotypes`
: This is an alternative output option for `fgl2haplo`

  `fgl2haplo` can output its haplotypes in MORGAN marker data file format, so that they can be more easily used in MORGAN downstream analyses. In this case the individual names from the `simpop_fgl` file are output as the first item in each line of genotypes.

`output with spaces`
: This statement can be used to insert spaces between the alleles in the output file of haplotypes produced by `fgl2haplo`. The file is twice as large, but it may be easier for input to downstream analysis programs. The default is absence of spaces.

`output haplotype labels`
: This statement implies the "output with spaces" option for haplotype output. If haplotype labels are requested, and if haplotypes are to be output as rows, then each row will include as the first item the ID name of the individual to whom the haplotype belongs. If labels are requested, and if haplotypes are output as columns, then the first line of the output will contain the ID names of the individual to whom the corresponding haplotype column belongs.

`output four-gamete state order jacquard`
: In the case of four gametes, the user may select to output states in the traditional "Jacquard" order: 1111, 1122, 1112, 1121, 1123, 1211, 1222, 1233, 1212, 1221, 1213, 1231, 1223, 1232, 1234. If the gametes are of a pair of individuals, and the data are analyzed as unphased, the output state-probabilities will be reduced to nine, in the ordering: 1111, 1122, 1112+1121, 1123, 1211+1222, 1233, 1212+1221, 1213+1231+1223+1232, 1234.

See [Concept Index], page 151, for; state ordering: Jacquard state ordering: lexicographic

### 12.10.4 ibd_haplo parameter statements

The following statements are specific to `ibd_haplo`, or have a particular role in this program. Note that a number of the statements apply only when the proband gamete set consists of the four gametes of a pair of individuals.

`select ([partially] phased | unphased) data`
: The "select ... data" statement is used to inform "ibd_haplo" whether to handle the data as phased data, unphased data or partially phased data. If the data are phased there is no restriction on whether proband gametes are related to each other or not. If the data are unphased or partially phased it is necessary that the proband gametes are pairs of haplotypes, each pair belonging to a whole individual.

`select [2009 | 2011] state transition matrix`
: There are two different state transition matrices implemented in the `ibd_haplo` program. The user must specify which transition matrix to use for the analysis: see Section 12.8 [Sample parameter files for ibd_haplo], page 127. Note that the 2009 matrix is applicable only to sets of four gametes.

`set transition matrix null fraction` *`X`*
: This statement sets a parameter that modifies the transition matrix to allow for transitions that can not occur under the base transition matrices. The argument, X, is a real number greater than or equal to 0.0 and less than or equal to 1.0.

`set genotyping error rate` *E*

This statement sets the genotyping error rate to be used by "ibd_haplo". The value of R is a real number greater than or equal to 0.0 and less than 1.0. The value 0.01 would be a typical value for R.

`set population kinship` *X*

This statement sets the prior population kinship parameter to be used by "ibd_haplo" to X, where X is a real number greater than 0.0 and less than 1.0. Typically in small populations a value from 0.01 to 0.05 might be reasonable.

`set kinship change rate` *X*

This statement sets the kinship change rate parameter for IBD. This is the total change rate per centiMorgan. It should be a real number greater than 0.0. It is approximately the prior for the inverse of an IBD segment length in centiMorgans between any pair of haplotypes. However, a smaller value than the typical expected length generally works better.

`set [scoreset` *N*`] proband gametes` *N1 K1 N2 K2* `...`

One or more scoring sets may be given, where a scoring set consists of two or more haplotypes. If there is more than one set, each set is assigned a number 1 or greater. The maximum number of haplotypes in each set is limited to 10, due to computer memory considerations. Pairs of names and meiosis indicators are given, with 0 indicating maternal inheritance, 1 indicating paternal inheritance. At least one proband gametes score set must be specified when running `ibd_haplo`.

This statement is also used by `fgl2ibd`

`set proband gametes all individual pairs`

If this statement is used, then the `ibd_haplo` program will set up scoresets of 4 gametes for every pair of observed individuals. Typically the user will have unphased genotypes for this purpose, although typically the data may be specified an phased or as unphased.

This statement is also used by `fgl2ibd`

`output four-gamete state order jacquard`

In the case of four gametes, the user may select to output states in the traditional "Jacquard" order: 1111, 1122, 1112, 1121, 1123, 1211, 1222, 1233, 1212, 1221, 1213, 1231, 1223, 1232, 1234. If the gametes are of a pair of individuals, and the data are analyzed as unphased, the output state-probabilities will be reduced to nine, in the ordering: 1111, 1122, 1112+1121, 1123, 1211+1222, 1233, 1212+1221, 1213+1231+1223+1232, 1234.

`output scores at markers` *I1 I2* `....`

The HMM algorithms of `ibd_haplo` require that all markers by used in the computation of the marker-based conditional probabilities of *ibd* states. However, subsequent analyses may not require computation at every marker location. Thus, to limit output files, it is possible to request the state probabilities to be output only at a subset of the markers.

# 13 Polygenic Modeling of Quantitative Traits by EM Algorithm

See [Concept Index], page 151, for: polygenic model, PolyEM, EM algorithm, quantitative trait.

## 13.1 Introduction to PolyEM programs

See [References], page 149, for details of the cited papers.

PolyEM is a set of programs to evaluate the likelihood and compute MLEs for polygenic models of quantitative traits by EM algorithms. The original versions of these programs were based on the work described in [TS90] and [TS92].

There are four main programs whose features are summarized below:

- `univar`: This program fits a univariate trait model. It is primarily for test purposes. The likelihood is computed by three methods including classical pedigree polygenic peeling [ES71], which does not extend to looped pedigrees, by direct inversion of the covariance matrix, which does not extend to large pedigrees, and by a general matrix elimination peeling method which is the method used by the other `PolyEM` programs.
- `unibig`: An extension of `univar` to big pedigrees that implements more efficient methods to compute the polygenic likelihood on large looped pedigrees.
- `bivar`: An extension of `unibig` for bivariate traits.
- `multivar`: An extension of `bivar` for multivariate traits.

All programs can work with looped pedigrees. The exception is that looped pedigrees cannot be used for the polygenic peeling algorithm in `univar`. The other programs do not use polygenic peeling to evaluate the likelihood.

Only examples and statement references for `multivar` are given since it has the most complete features. The statements for other programs are similar with some exceptions. For example, any statements with between-trait covariances do not apply to `univar` or `unibig`, since these deal only with a single trait.

See [Concept Index], page 151, for: PolyEM introduction, `univar`, `unibig`, `bivar`, `multivar`.

## 13.2 Sample `multivar` parameter file

The example pedigree file `polyem.ped` for the PolyEM programs is a 90-member pedigree consisting of two 45-member components. The format is similar to `ped73.ped`, which was used in most of the previous examples.

The first three entries in each line consist of the individual's name, father's name and mother's name. Integers starting with the fourth column (usually gender) can be fixed effects (gender, age class, etc.) or discrete phenotypes.

For quantitative traits, real numbers follow the names and integers. These real numbers represent trait measurements. Missing values are coded with integer part '999', such as 999.5 in the following example.

Here is part of the pedigree file `polyem.ped`. This file can be found in the `PolyEM` subdirectory of `MORGAN_Examples`.

```
input pedigree size 90
```

```
 input pedigree record names 3 integers 3 reals 2
****************************************
    1     0     0     1     1     0    0.0246   -1.0125
    2     0     0     2     1     0   -0.5978    1.5963
    3     0     0     1     1     0   -0.8124    0.5662
    4     0     0     2     1     0    0.4334    1.7721
    5     1     2     1     1     0    0.1802   -1.4672
    6     1     2     1     1     0   -1.7557    0.8091
    7     3     4     2     1     0    999.5     999.5
    8     3     4     2     1     0    1.9128    0.9780
    9     0     0     2     1     0    0.9530    2.3473
    ...
```

Below is the example `multivar` parameter file, `polyem.par`.

```
input pedigree file polyem.ped

select traits 1 2
set trait 2 effects 1 2

start residual covariance  -0.09
start additive covariance  -0.0017
start residual variance     1.10   0.65
start additive variance     0.037  0.0288

fit residual covariance
    1

fit additive covariance
    1

fit environmental model

output spacing 20 EM iterations
limit EM iterations 200
```

`multivar` can fit a polygenic model with one to five traits, which can be modeled as dependent and/or independent. One '`select traits`' statement must be given. The number of integer values entered as arguments to the statement must be the number of quantitative traits expected by the program being run. For example, for programs `univar` and `unibig`, the '`select traits`' statement must have a single integer argument. When running `bivar`, two integer arguments must be given. For `multivar`, one to five integer arguments must be given in the '`select traits`' statement, for the one to five quantitative traits selected. Unlike other MORGAN programs, the trait numbers correspond to the column number of the reals in the pedigree file. In the example, the statement '`select traits 1 2`' indicates that the first two column of real numbers contain the trait data to be analyzed.

The statement '`set trait 2 effects 1 2`' indicates that the second column of real numbers is to be modeled with two *fixed effects* (also called *covariates*). The integers give the location of the fixed effects (covariates) *starting with column 4* in the pedigree file. In this example, the fixed effects are to be found in columns 4 and 5. **Important:** a fixed effect location of '`1`' indicates that the effect value will be found in column 4 (*after* the 3 name columns). The most commonly modeled fixed effect is gender, which, if present, resides in column 4 of a MORGAN pedigree file.

The statement '`start trait mean`' allows the user to specify the starting trait mean for a selected trait. Since no '`start trait mean`' statement is included after either of the '`select trait`' statements, both of the initial means are computed by the PolyEm program. Similarly, one may specify the initial values for each effect with the statement '`start trait I effect M X1 X2 ...`', where '`I`' is the trait number, '`M`' is the effect number and '`Xi`' is the starting value of the *i*th level (i = 1, 2, 3, ...). These starting values represent deviations from the global mean. The starting values are normalized so that their weighted sum is zero (weighted by the number of individuals in that level). When using the '`start trait I effect M X1 X2 ...`' it is important to know that if more levels are present in the column of numbers corresponding to the trait '`I`' in the pedigree file than are specified in the '`start trait`' statement, PolyEm programs will compute the starting value(s) for these additional levels. Since the program will not issue a warning or error message, it is important to always check the output to confirm that the number of levels present in the file was as intended. Since the '`start trait I effect M X1 X2 ...`' statement is not included in this example, the PolyEm program will compute the initial values of the effect.

Initial values for additive and residual variances and covariances are specified in the next four statements. These statements are required. With the variance statements, the number of arguments must be the same as the number of traits selected and must be in order of increasing trait number. With the covariance statements, the number of arguments must be the same as the number of pairs of traits selected. See Section 13.4.2 [multivar segregation model parameters], page 139, for discussion of the ordering of these arguments.

`multivar` can also fit a purely environmental model with no genetic component. The '`fit environmental model`' statement tells `multivar` to fit a purely environmental model, with no genetic variance. This *null hypothesis* model is produced in addition to the genetic/environmental model.

The final two statements specify the number of EM iterations and how often the EM estimates are to be printed out.

Note that one has the opportunity to provide predetermined eigenvalues of the G-matrix of observed individuals. The '`set eigenvalues`' statement is used to specify the eigenvalues, with the number of values equal to the number of observed individuals. If desired, the eigenvalues can be provided through an input file accessed with a '`input eigenvalue file`' statement in the parameter file, or through the command line (see Section 13.4.3 [multivar computational parameters], page 140).

See [Concept Index], page 151, for: `multivar` sample parameter file, missing quantitative trait data.

## 13.3 Running multivar example and sample output

The command to run `multivar` (`unibig` and `bivar` have the same set of options) is:

```
./multivar parfile [ped pedfile] [eigen eigenfile]
```

where *parfile* is the name of the parameter file and is required. *pedfile* overrides the '`input pedigree file`' statement, and *eigenfile* overrides the '`input eigenvalue file`' statement in the parameter file.

Under the subdirectory `PolyEM/`, run the example by typing:

```
./multivar polyem.par
```

Toward the end of the `multivar` output, are the parameter estimates and the log-likelihood from the last iteration of the EM algorithm. If you chose to fit a null (purely environmental) model, using the '`fit environmental model`' statement, those parameter estimates and log-likelihood are also given. A likelihood ratio test can then be performed, with test statistic equal to the absolute value of 2 times the difference between the log-likelihoods of the two models. A conservative test is provided by comparing the test statistic to a chi-squared distribution, with the degrees of freedom being the difference in the numbers of estimated parameters between these two models. Note that in these `Polyem` programs, model log-likelihoods are in base $e$ rather than the usual lod score base-10 convention.

```
iteration #201:

   additive variance estimates (traits 1, 2)
       0.816    0.037
   covariances
       0.138

   residual variance estimates (traits 1, 2)
       0.223    0.610
   covariances
      -0.239

   trait 1
      overall mean      -0.063
   trait 2
      overall mean       1.780
      fixed effect  1   -0.717    0.546
      fixed effect  2   -1.008   -0.552    1.167

current log-likelihood = -183.098

estimates of environmental model

   residual variance estimates (traits 1, 2)
       1.136    0.642
   covariances
      -0.102

   trait 1
      overall mean       0.062
   trait 2
      overall mean       1.801
```

```
            fixed effect  1   -0.773    0.589
            fixed effect  2   -1.010   -0.553    1.170

      environmental model log-likelihood = -197.799
```

In this example, we see that the fitted genetic model has a very significantly larger log-likelihood: 2(-183.098 + 197.799) = 29.4, with 3 extra genetic variance parameters fitted. The estimates of the fixed effects are little altered by fitting the genetic model but for trait 1 the additive genetic variance is large relative to the residual variance, indicating a strong genetic component. Note that, for each trait, the sum of the additive and residual variances from the genetic model is close to the residual variance in the environmental model.

See [Concept Index], page 151, for: running `multivar` example, `multivar` sample output.

## 13.4 `multivar` statements

See [Concept Index], page 151, for: PolyEM statements, `multivar` statements.

### 13.4.1 `multivar` computing requests

`select traits` *`I1...`*

One '`select trait`' statement is required, and must list the traits to be modeled. Up to five traits are allowed for `multivar`. The trait number, *I*, corresponds to the column of real numbers in the pedigree file, with the first column of real numbers being trait 1, the second column trait 2 and so on.

`set trait` *`I`* `effects` *`M1...`*

*M1...* are the fixed effects to be modeled for a specified trait *I*. They are the integer columns in which they appear in the pedigree file. That is the columns after the three names, so that fixed effect '`1`' is in the 4th column (usually *gender*), effect '`2`' is in column 5, and so on.

See [Concept Index], page 151, for: `multivar` computing requests.

### 13.4.2 `multivar` segregation model parameters

`start trait` *`I`* `mean` *`X`*

There is one statement per trait, specifying the starting value for the mean trait values. The PolyEM programs will compute the initial values if not given.

`start trait` *`I`* `effect` *`M X1 X2`* `...`

Starting values for the fixed effect levels for the traits are computed, unless specified in this statement.

`start additive variances` *`X1 X2`* `...`

The starting values for the variances of the traits are required. The number of values must be the same as the number of traits selected, in the order of increasing trait number.

`start residual variances` *`X1 X2`* `...`

Starting values for residual variances are also required.

`start additive (covariances | correlations)` *`X12`* `...`
: Starting values for the covariances (or correlations) between the traits are required. They are given the order: *X12*, *X13*, ..., *X1n*, *X23*, ..., *X2n*, ..., where *Xij* is the covariance for the $i$th and $j$th selected traits from the pedigree file.

`start residual (covariances | correlations)` *`X12`* `...`
: See the '`start additive...`' statements.

See [Concept Index], page 151, for: `multivar` segregation model parameters.

### 13.4.3 `multivar` computational parameters

`fit additive (covariances | correlations)` *`X12`* `...`
: This statement specifies which covariances to be estimated and which to be fixed at 0. The order of values is the same as the '`start additive covariances`' statement with '1' indicating a covariance to be fit and '0' a covariance to be fixed.

  Note that if trait 1 is correlated with trait 3, and so is trait 2 with trait 3, the correlation between 1 and 2 cannot be zero. So we have to be a bit careful in specifying the correlation structure.

`fit residual (covariances | correlations)` *`X12`* `...`
: Similar statement for residual covariances.

`set eigenvalues` *`X1 X2`* `...`
: Optional. This statement is used to provide predetermined eigenvalues of the G-matrix of observed individuals, with the number of values the same as the number of observed individuals.

`input eigenvalue file` *`eigenfile`*
: Optional. If present, it overrides the '`set eigenvalues`' statements.

`limit breeding iterations` *`I`*
: This statement specifies the maximum number of breeding–values iterations. The default number currently is 20.

`set breeding convergence` *`X`*
: This statement specifies the convergence criterion for breeding–values iterations. The default number is currently is 1.0e-8.

`limit EM iterations` *`I`*
: This statement specifies the number of EM iterations. The default number presently is 200. There is no option to specify convergence criterion. If convergence has not been achieved, the final estimates can be used as starting values to rerun the program.

See [Concept Index], page 151, for: `multivar` computational parameters, observed individuals.

### 13.4.4 `multivar` computational options

`compute eigenvalues`
: If this statement is present, the values given in either the *eigenfile* or the statement '`set eigenvalues`' are ignored and the eigenvalues are computed by the program. This is the default action if no eigenvalues are given.

`use (full | partitioned) EM`
: Use this statement to choose between two iterative procedures in maximum likelihood estimates. With the '`full EM`' option, the fixed effects, additive and residual variance and covariance are simultaneously updated. This is the default action.

  With the '`partitioned EM`' option, the maximization step is partitioned into two parts. The first part is to maximize the likelihood over additive and residual variances/covariances; the second part over residual variances/covariances and fixed effects. The expectation step is run after each part. Partitioned EM takes more computer time.

`fit environmental model`
: This statement asks a purely environmental model with no genetic variances to be fit, in additional to the genetic/environmental model.

See [Concept Index], page 151, for: `multivar` computational options.

### 13.4.5 `multivar` output options

`output statistics (covariances | correlations)`
: By default, covariances are printed out.

`output final adjusted phenotypes`
: If this option is specified, trait values adjusted for all fixed effects are computed and output.

`output spacing` *I* `EM iterations`
: This statement requests a print out of the EM estimates every $I$th iteration. The default number is defined in the program header file.

`check gmatrix`
: This statement requests a print out of the G matrix for observed individuals and quit without doing the likelihood computation.

`check ginverse`
: This statement requests a print out of the G inverse matrix and the program quits unless '`check eigenvalues`' has also been specified.

`check eigenvalues`
: This statement requests the program to print out of the eigenvalues, whether computed or input, and then to quit. These eigenvalues can then be used as input in subsequent runs.

`check eigenvalue computation`
: This statements causes some comments to be printed by the function that computes the eigenvalues.

`check trace`

This statements requests the trace of the G–inverse matrix to be printed.

See [Concept Index], page 151, for: `multivar` output options.

# 14 Estimating Genetic Maps from Marker Data

See [Concept Index], page 151, for: genetic map estimation.

## 14.1 Introduction to `lm_map`

See [References], page 149, for details of the cited papers.

The program `lm_map` finds the maximum likelihood estimate (MLE) of the marker map, estimates the (statistical not Monte Carlo) variance of the MLE, and tests hypotheses about the true map. All inference is based on the analysis of multilocus marker data obtained from some (possibly all) members of a set of independent families (pedigree components).

To find the MLE, lm_map uses either Monte Carlo expectation-maximization (MCEM) or a hybrid of MCEM and stochastic approximation (SA). In either case, the user must supply an initial map estimate, and an initial Monte Carlo (MC) sample size for the MCEM algorithm. The MCEM sample size is automatically increased with each successive step of the algorithm, and only a small number of MCEM steps are needed to estimate the MLE. If the hybrid option is chosen, `lm_map` uses the MCEM estimate to seed the SA algorithm. Then, a relatively large number of SA steps are used to estimate the MLE with greater precision.

Once the MLE is obtained, a long Markov chain is used to estimate the variance of the MLE. Finally, a slight adaptation of the MC likelihood ratio formula is used to estimate the likelihood ratio test (LRT) statistics for testing the simple and/or composite null hypotheses. For more details, see [ST06].

See [Concept Index], page 151, for: `lm_map` introduction.

## 14.2 Sample `lm_map` parameter file

The two sample parameter files for `lm_map` can be found in the directory `MORGAN_Examples/Map`. The two files are `map_G.par` and `map_P.par`, along with the corresponding marker data files `map_G.markers` and `map_P.markers`. Thus there are two examples, one for genotypic markers (G) and one for phenotypic markers (P). '`G`' denotes that marker genotypes are observed without error. '`P`' denotes the possibility of error, so that the observed marker phenotype is not the same as the underlying true marker genotype. This example uses the pedigree file `map.ped`, but different marker data files depending on the choice of '`P`' or '`G`'.

`map_G.par` and `map_P.par` have the following statements in common:

```
input pedigree file './map.ped'
input marker data file './map_G.markers'  # or './map_P.markers' for 'map_P.par'

select all markers
set marker 1 2 3 allele freqs  .2 .2 .2 .2 .2
set marker names DS123 DS456 DS789

map gender F marker recomb fract .18 .18  # true F map (cM): 20 20
map gender M marker recomb fract .08 .08  # true M map (cM): 10 10
```

```
limit recomb fracts .001

use sequential imputation for setup
use 100 sequential imputation realizations for setup
set burn-in iterations 100
sample by scan
set L-sampler probability .8
set MC iterations 50   # The initial number of MCMC scans per step
limit EM iterations 10   # The total number of MCEM steps
```

As seen in previous examples, the '`select all markers`' statement instructs the program to use all markers on the chromosome for computation. The alternative is to use only selected markers for computation, which can be achieved by using the '`select markers`' statement (see Section 9.8.1 [Autozyg computing requests], page 74). The '`set marker 1 2 3 allele freqs .2 .2 .2 .2`' statement specifies the marker allele frequencies for markers 1, 2, and 3. This statement, as constructed, requires markers 1, 2, and 3 to each have five alleles with frequencies of 0.2 for each allele. If the number of alleles per marker varies from marker to marker, or if the allele frequencies vary from marker to marker, a separate '`set marker freqs`' statement is needed for each marker (see Section 6.5.3 [markerdrop population model parameters], page 44). The '`set marker names`' statement overrides the default behavior, which labels markers consecutively: marker-1, marker-2, etc.

The two '`map gender marker recomb fract`' statements specify the marker map in terms of recombination fractions. This is the initial starting estimate of the map.

The '`limit recomb fracts 0.001`' statement is optional and places lower and upper bounds on the estimated recombination fractions of the map. For markers that are separated by little or no recombination, the MCEM algorithm may yield estimated recombination fractions of zero which could lead to a severe bias in the results. As a safeguard against such events, this statement places a lower bound 0.001 and an upper bound 0.5 - 0.001 on the estimated recombination fractions of the map.

The statement '`use sequential imputation for setup`' instructs `lm_map` to initialize the set of maternal and paternal meiosis indicators for all members of the pedigree who are not founders; this is done prior to the Monte Carlo simulation. The default behavior is specified in this statement, with the alternative being to '`use locus-by-locus sampling for setup`'. The statement '`use 100 sequential imputation realizations for setup`' is optional and modifies the default behavior for setup by sequential imputation (which is 10% of the MC iterations). The next three lines in the parameter files contain statements introduced in the `Autozyg` examples of this tutorial. For explanation of '`set burn-in iterations`', '`sample by scan`', and '`set L-sampler probability`' see Section 9.8.8 [Autozyg MCMC parameters and options], page 79. The statement '`set MC iterations 50`' indicates how many MC iterations are to be performed at each EM iteration. The statement '`limit EM iterations`' was introduced in the `multivar` example and puts an upper bound on the number of MCEM iterations.

Now we'll take a look at the remaining statements in `map_G.par`:

```
output maps gender averaged specific
set map estimation model with no mistyping
set EM convergence .01
```

```
use MCEM and SA for maximization
set SA curvature iterations 10
set SA ascent iterations 10
set SA gradient iterations 10
set SA convergence .001
```

The '`output maps gender averaged specific`' statement specifies the type of map to be estimated by `lm_map`. In this example, the default behavior is specified, which instructs `lm_map` to automatically compute the likelihood ratio test statistic for testing the null hypothesis of a sex-averaged map. The statement '`set map estimation model with no mistyping`' instructs `lm_map` to assume that the genotypes are observed without error. The '`set EM convergence`' statement instructs `lm_map` to stop the MCEM algorithm if all recombination fraction updates are within 0.01 of their previous values.

The statement '`use MCEM and SA for maximization`' in `map_G.par` instructs `lm_map` to attempt to refine its MCEM-based estimate of the MLE by performing additional SA steps. The alternative is to '`use MCEM only for maximization`', with no further refining. There are several statements that allow additional control of the SA algorithm. First, an estimate of the curvature of the likelihood is needed to initiate the SA algorithm. The statement '`set SA curvature iterations 10`' instructs `lm_map` to use at least 10 MCMC realizations to estimate the curvature of the likelihood. Also, `lm_map` will not initiate the SA algorithm with a step that decreases likelihood. So, when the SA algorithm is used for refining the likelihood estimate, the statement '`set SA ascent iterations 10`' instructs `lm_map` to use at least 10 MCMC realizations to determine whether a proposed first step increases the likelihood. The SA algorithm also requires an estimate of the gradient of the likelihood at each SA step. The statement '`set SA gradient iterations 10`' instructs `lm_map` to use at least 10 MCMC realizations to estimate the gradient of the likelihood. Finally, the map estimate obtained from the final step of the MCEM algorithm is used to seed the SA algorithm. The '`set SA convergence 0.001`' statement instructs `lm_map` to terminate the SA algorithm when the absolute change in successive map estimates is less than 0.001 for each recombination fraction in the map.

The file `map_P.par` shows some different Monte Carlo and estimation options. Here are the remaining statements in that file:

```
output maps gender averaged
set map estimation model with mistyping
set genotyping error rate .02
use MCEM only for maximization
```

In this parameter file, a gender averaged map is specified by using the '`output maps gender averaged`' statement. Unlike in the previous parameter file, `map_P.par` does not assume the genotypes are recorded without error; this is indicated by the statement '`set map estimation model with mistyping`'. When '`with mistyping`' is chosen, one has the option of specifying an estimate of the error rate with the statement '`set genotyping error rate` *E*'. In this example, the error rate is set at 0.02. Finally, the statement '`use MCEM only for maximization`' instructs `lm_map` not to use the SA algorithm to further refine the MCEM-based estimate of the MLE. Since the SA algorithm will not be used, none of the '`SA`' statements are used in `map_P.par`.

See [Concept Index], page 151, for: sample parameter file for `lm_map`.

## 14.3 Running `lm_map` with genotypic data

Run the genotypic example in the `Map` subdirectory of the `MORGAN-examples` directory with the following command

```
./lm_map map_G.par
```

The `lm_map` program is one of the more computationally intensive MORGAN programs. Even running this small example takes about 30 seconds to run (depending on the computer used, of course). Again, different random seeds will result in different outputs with each run.

Here is the output from one of the runs: The maximum likelihood estimates (MLEs) of marker map recombination frequencies are given for each marker interval and for male and female meioses. Also given is the estimated variance-covariance matrix of the MLEs. Of course, the MLE will not be identical to the true parameter value, but the variance-covariance matrix gives an estimate of the precision. The '`effective number of meioses`' is also a measure of this precision, giving the number of fully informative meioses required for the same precision of the MLEs.

```
          MAXIMUM LIKELIHOOD ESTIMATES

       Interval    Female (RF)         Male (RF)
       --------    -----------         ---------
          1           0.2231            0.0264
          2           0.2745            0.0801

      ESTIMATED VARIANCE OF SEX-SPECIFIC MAP [F1,M1,F2,M2,... x F1,M1,F2,M2,...]

         0.004402  -0.000034  -0.000422  -0.000040
        -0.000034   0.000621   0.000075  -0.000025
        -0.000422   0.000075   0.006546  -0.000136
        -0.000040  -0.000025  -0.000136   0.001482

      EFFECTIVE NUMBER OF MEIOSES

      Interval  Female      Male
     --------  --------    -------
         1:      40          42
         2:      31          50
```

See [Concept Index], page 151, for: running `lm_map` with genotypic data, `lm_map` sample output for genotypic data.

## 14.4 Running `lm_map` with phenotypic data

```
./lm_map map_P.par
```

Running this example takes a noticeable amount of time. Given are the MLEs of the sex-averaged recombination frequency in each of the two marker intervals and of the mistyping (error) rate. Also given is the estimated variance-covariance matrix of these MLEs and the effective number of meioses (see the previous section).

```
             MAXIMUM LIKELIHOOD ESTIMATES

        Interval        Sex-Averaged (RF)
        --------        ----------------
          1                  0.1510
          2                  0.1787

             MISTYPING RATE: 1.479401%


       ESTIMATED VARIANCE OF (MAP,MISTYPING RATE) SEX-AVERAGED
      ------------------------------------------------------
          0.001432  -0.000482   0.000007
         -0.000482   0.001517  -0.000016
          0.000007  -0.000016   0.000072
```

Following this section, there is a table of the estimated error probability for each individual at each marker. From your output you should see that the program detects errors in individual 32 and individual 49 for marker-3 and in individual 90 for marker-1. Some other instances of data with low (non-error) probability also show non-zero estimated probability of error. The exact values of these probabilities will depend on the random seeds used in the run.

See [Concept Index], page 151, for: running `lm_map` with phenotypic data, `lm_map` sample output for phenotypic data.

## 14.5 `lm_map` statements

`limit recombination fractions L`
: This statement is optional and places lower and upper bounds on the estimated recombination fractions of the map. For markers that are separated by little or no recombination, the MCEM algorithm may yield estimated recombination fractions of zero which could lead to a severe bias in the results. As a safeguard against such events, this statement places a lower bound L and an upper bound 0.5 - L on the estimated recombination fractions of the map.

`output maps gender [averaged] [specific]`
: This statement specifies the type of map to be estimated. The default behavior is to select both options, which instructs lm_map to automatically compute the likelihood ratio test statistic for testing the null hypothesis of a sex-averaged map.

`use MCEM and SA for maximization`
: If the statement '`use MCEM only for maximization`' is replaced by this statement, `lm_map` will attempt to refine its MCEM based estimate of the MLE by performing additional SA steps.

`set EM convergence` *`X`*
: The MCEM algorithm is used to find a suitable starting value for the SA algorithm. The MCEM algorithm terminates when the percent change in successive

parameter estimates is less than $X$. The default value of $X$ is 0.2: smaller values may substantially increase the total CPU time.

`set genotyping error rate` *E*
: When the statement '`set map estimation with mistyping`' is used, the genotype observations are assumed to have an associated error rate. This statement allows for the specification of the '`mistyping`' rate.

`set SA curvature iterations` *I*
: An estimate of the curvature of the likelihood is needed to initiate the SA algorithm. This statement tells `lm_map` to use at least $I$ MCMC realizations to estimate the curvature of the likelihood. The curvature is only estimated once.

`set SA ascent iterations` *I*
: `lm_map` will not initiate the SA algorithm with a step that decreases the likelihood. This statement tells `lm_map` to use at least $I$ MCMC realizations to determine whether a proposed first step increases the likelihood.

`set SA gradient iterations` *I*
: If SA is initiated, this tells `lm_map` to use at least $I$ MCMC realizations to estimate the gradient of the likelihood. An estimate of the gradient is needed for each SA step.

`set SA convergence` *R*
: The SA algorithm is terminated, if all recombination fraction updates are within $R$ of their previous values. In addition, the maximum possible runtime for the SA algorithm is proportional to the total runtime of the MCEM algorithm.

`set map estimation (with | with no) mistyping`
: This statement can be used to specify whether or not errors were made during the observation of genotype. If '`with no`' is selected, the genotypes are assumed to have been observed without error. If '`with`' is selected, the genotype observations are assumed to have some error associated with them, which can be specified using the '`set genotyping error rate`' statement.

`set LRT statistics iterations` *I*
: This statement tells `lm_map` to use at least $I$ MCMC realizations to estimate the LRT statistics. If only one option is used in '`output maps gender ...`', then the estimated LRT statistic compares the MLE to the initial map. Otherwise, two LRT statistics are estimated. The first compares the MLE of the sex-averaged map to the initial sex-averaged map, while the second compares the MLE of the sex-specific map to the MLE of sex-averaged map.

`compute estimates` *I* `times`
: This statement tells `lm_map` to conduct its entire analysis $I$ times, and to report the map with the highest likelihood. While this statement offers some protection against convergence to local modes, the default value is 1.

See [Concept Index], page 151, for: `lm_map` statements.

# References

- [Bas08] Basu, S., Di, Y., and Thompson, E. A. (2008), "Exact trait-model-free tests for linkage detection in pedigrees," *Annals of Human Genetics* **72**, 672–682.
- [Bas10] Basu, S., Stephens, M., Pankow J. S., and Thompson, E. A. (2010), "A likelihood-based trait-model-free approach for linkage detection of binary trait," *Biometrics* **66**, 205–213.
- [Bau72] Baum, L E. (1972), "An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes," *in* O. Shisha, ed., 'Inequalities-III; Proceedings of the Third Symposium on Inequalities. University of California Los Angeles, 1969', Academic Press, New York, pp. 1–8.
- [BGZT12] Brown, M. D., Glazner, C. G., Zheng, C., and Thompson, E. A. (2012), "Inferring coancestry in population samples in the presence of linkage disequilibrium." *Genetics*, **190**, 1447–1460.
- [Browning06] Browning, S. R. (2006) "Multilocus association mapping using variable-length Markov chains" *American Journal of Human Genetics*, **78:** 903–913.
- [Don83] Donnelly, K. P. (1983), "The probability that related individuals share some section of genome identical by descent" *Theoretical Population Biology*, **23**, 34–63.
- [DT09] Di, Y. and Thompson, E. A. (2009), "Conditional tests for localizing trait genes," *Human Heredity* **68**, 139–150.
- [ES71] Elston, R. C. and Stewart, J. (1971), "A general model for the analysis of pedigree data," *Human Heredity* **21**, 523–542.
- [GT03] George, A. W. and Thompson, E. A. (2003) "Discovering disease genes: Multipoint linkage analysis via a new Markov Chain Monte Carlo approach," *Statistical Science* **18**, 515–531.
- [GT15] Glazner, C. G, and Thompson, E. A. [2015] "Pedigree-free descent-based gene mapping from population samples." *Human Heredity* **80**, 21–35.
- [GWT05] George, A. W., Wijsman, E. M. and Thompson, E. A. (2005) MCMC Multilocus Lod Scores: Application of a New Approach. *Human Heredity* **59:** 98–108.
- [GYO96] Goddard, K. A., Yu, C. E., Oshima, J., Miki, T., Nakura, J., Piussan, C., Martin G. M. et. al. (1996), "Toward localization of the Werner syndrome gene by linkage disequilibrium and ancestral haplotyping: lessons learned from analysis of 35 chromosome 8p11.1-21.1 markers," *American Journal of Human Genetics*, **58:** 1286–1302.
- [Hal19] Haldane, J. B. S. (1919), "The combination of linkage values and the calculation of distances between the loci of linked factors" *Journal of Genetics*, **8:** 229–309. '
- [He97] Heath, S.C. (1997). Markov chain Monte Carlo segregation and linkage analysis for oligogenic models. *American Journal of Human Genetics*, **61:** 748-760.
- [KT13] Koepke, H.A. and Thompson, E.A. (2013). Efficient identification of equivalences in dynamic graphs and pedigree structures. Journal of Computational Biology, 20: 551-570.
- [LG87] Lander, E. S. and Green, P. (1987), "Construction of multilocus genetic linkage maps in humans," *Proceedings of the National Academy of Sciences (USA)* **84**, 2363–2367.

– [Sie05] Sieh, W., Basu, S., Fu, A. Q., Rothstein, J. H., Scheet, P. A., Stewart, W. C. L., Sung, Y. J., Thompson, E. A., and Wijsman, E. M. (2005), "Comparison of marker types and map assumptions using Markov chain Monte Carlo-based linkage analysis of COGA data," *BMC Genetics* **6** (Suppl 1), S11.

– [ST06] Stewart, W. C. L. and Thompson, E. A. (2006), "Improving estimates of genetic maps: A maximum likelihood approach," *Biometrics* **62**, 728–734.

– [STW07] Sung, Y. J., Thompson E. A., and Wijsman E. M. (2007), "MCMC-based linkage analysis for complex traits on general pedigrees: multipoint analysis with a two-locus model and a polygenic component," *Genetic Epidemiology* **31**, 103–114.

– [SW07] Sung, Y. J. and Wijsman, E. M. (2007), "Accounting for epistasis in linkage analysis of general pedigrees," *Human Heredity* **63**, 114–152.

– [TG07] Thompson, E. A., and Geyer, C. J. (2007), "Fuzzy p-values in latent variable problems," *Biometrika* **90**, 49–60.

– [Tho76] Thompson, E. A. (1976), "Peeling programs for zero-loop pedigrees," Technical Report No. 5, Department of Medical Biophysics and Computing, University of Utah.

– [Tho77] Thompson, E. A. (1977), "Peeling programs for pedigrees of arbitrary complexity," Technical Report No. 6, Department of Medical Biophysics and Computing, University of Utah.

– [Tho00] Thompson, E. A. (2000), "Statistical Inference from Genetic Data on Pedigrees," *in* 'NSF-CBMS Regional Conference Series in Probability and Statistics,' Vol. 6, Institute of Mathematical Sciences, Beachwood, Ohio.

– [Tho08a] Thompson, E. A. (2008), "Uncertainty in inheritance: assessing linkage evidence," *in* 'Joint Statistical Meetings Proceedings,' Salt Lake City 2007, pp. 3751–3758.

– [Tho08b] Thompson, E. A. (2008), "The IBD process along four chromosomes." *Theoretical Population Biology,* **73:** 369–373.

– [Tho11] Thompson, E. A. (2011), "The structure of genetic linkage data: from LIPED to 1M SNPs," *Human Heredity,* **71:** 86–96.

– [TH99] Thompson, E. A. and Heath, S. C. (1999). Estimation of conditional multilocus gene identity among relatives. Statistics in Molecular Biology and Genetics, Seillier-Moseiwitch F., (ed). IMS Lecture Note - Monograph Series, 33: 95-113.

– [TS90] Thompson, E. A. and Shaw, R. G. (1990). Pedigree analysis for quantitative traits: variance components without matrix inversion. Biometrics, 46: 399-413.

– [TS92] Thompson, E. A. and Shaw, R. G. (1992). Estimating polygenic models for multivariate data on large pedigrees. Genetics, 131: 971-978.

– [TT08] Tong, L. and Thompson, E. A., "Multilocus lod scores in large pedigrees: Combination of exact and approximate calculations," *Human Heredity* **65**, 142–153.

# Concept Index

## A

*a priori ibd* probabilities . . . . . 46
additive variance . . . . . 29, 31, 34
affected individuals . . . . . 79
age-of-onset penetrance specification . . . . . 94
age-of-onset penetrances . . . . . 78
allele frequencies . . . . . 34
allow statements . . . . . 12
Autozyg computational parameters . . . . . 77
Autozyg computing requests . . . . . 74
Autozyg file identification statements . . . . . 75
Autozyg mapping model parameters . . . . . 76
Autozyg MCMC parameters and options . . . . . 79
Autozyg output file description . . . . . 76
Autozyg pedigree file description . . . . . 76
Autozyg population model parameters . . . . . 77
Autozyg programs . . . . . 63
Autozyg statements . . . . . 74
autozygosity . . . . . 63

## B

base log likelihood . . . . . 101, 109
`bivar` . . . . . 135
block Gibbs sampler . . . . . 58
burn-in . . . . . 60, 61, 66

## C

`civil` introduction . . . . . 80
`civil` sample output . . . . . 85
`civil` statements . . . . . 87
Cleopatra pedigree . . . . . 50
command line options . . . . . 9
command syntax . . . . . 9
component . . . . . 18, 19, 21, 24
conditional *ibd* probabilities . . . . . 63
consistency (Mendelian) of marker data . . . . . 74
continuous trait specification . . . . . 94

## D

debug control . . . . . 13
dense markers . . . . . 37, 49
descent graph . . . . . 63
discrete trait . . . . . 45, 78
discrete trait specification . . . . . 93
display –GLUT runtime . . . . . 13
display options . . . . . 13
dummy statements . . . . . 13, 25
dummy variables . . . . . 13

## E

EM algorithm . . . . . 135
exact computation . . . . . 70
exact computations . . . . . 57
examples using pedigree file `ped73.ped` . . . . . 5
extra file . . . . . 11

## F

file names . . . . . 11
filetype codes . . . . . 9
founder genome labels . . . . 29, 31, 35, 42, 56, 70, 76
founder genome labels (FGL) . . . . . 37
fuzzy p-value . . . . . 73

## G

gender–specific maps . . . . . 33, 65, 91
gender: see also individuals –gender . . . . . 16
`genedrop` computational parameters . . . . . 35
`genedrop` computing requests . . . . . 33
`genedrop` introduction . . . . . 29
`genedrop` mapping model parameters . . . . . 33
`genedrop` output pedigree options . . . . . 35
`genedrop` output seed file . . . . . 36
`genedrop` population model parameters . . . . . 34
`genedrop` sample output . . . . . 32
`genedrop` sample parameter file . . . . . 30
`genedrop` statements . . . . . 33
genetic map estimation . . . . . 143
genetic model . . . . . 56
genotypic trait . . . . . 65, 78
genotypic trait specification . . . . . 93
`gl_auto` introduction . . . . . 63
`gl_auto` sample output . . . . . 71
`gl_auto` sample parameter file . . . . . 69
`gl_auto` statements . . . . . 74
`gl_lods` statements . . . . . 111

## H

Haldane map function –See also map function . . 33
HMM computations . . . . . 57
how to get the examples . . . . . 3
how to get the tutorial . . . . . 3

## I

*ibd* . . . . 22, 46, 56, 63
*ibd* graph . . . . 26, 63, 70, 101, 117, 123, 132
*ibd* measures . . . . 87
*ibd* pattern . . . . 46, 52, 68
*ibd*-based tests . . . . 80
`ibd_class` utility . . . . 26
`ibd_create` introduction . . . . 117
`ibd_haplo` introduction . . . . 117
`ibddrop` introduction . . . . 46
`ibddrop` sample output . . . . 51
`ibddrop` sample parameter file . . . . 47
`ibddrop` statements . . . . 54
IBDgraph library . . . . 26
identity by descent . . . . 46, 63
inbreeding coefficient . . . . 22, 25
incomplete penetrance . . . . 45, 78
incomplete penetrances . . . . 37
individuals –gender . . . . 16, 21
individuals –names . . . . 14
individuals file . . . . 11, 15, 19, 20, 101
inheritance indicators; See meiosis indicators . . . . 35
inheritance vector . . . . 56

## K

`kin` introduction . . . . 22
`kin` sample output . . . . 23
`kin` sample parameter file . . . . 22
`kin` statements . . . . 24
kinship coefficient . . . . 22, 25
Kosambi map function –See also
  map function . . . . 33

## L

L-sampler . . . . 58, 61, 66, 92
L-sampler probability . . . . 59
latent p-values . . . . 63, 83
liability class specification . . . . 94
liability classes . . . . 78
likelihood-ratio measures . . . . 87
limit overriding . . . . 12
limit statements . . . . 12
line length limit . . . . 10
LINKAGE package format . . . . 25
`lm_auto` introduction . . . . 63
`lm_auto` sample output . . . . 67
`lm_auto` sample parameter file . . . . 64
`lm_auto` statements . . . . 74
`lm_bayes` introduction . . . . 89
`lm_bayes` sample output . . . . 96
`lm_bayes` statements . . . . 111
`lm_ibdtests` introduction . . . . 80
`lm_ibdtests` sample output . . . . 82
`lm_ibdtests` statements . . . . 87
`lm_linkage` introduction . . . . 89
`lm_linkage` sample output . . . . 95
`lm_linkage` statements . . . . 111
`lm_map` introduction . . . . 143
`lm_map` sample output for genotypic data . . . . 146
`lm_map` sample output for phenotypic data . . . . 146
`lm_map` statements . . . . 147
`lm_pval` introduction . . . . 63
`lm_pval` sample output . . . . 73
`lm_pval` sample parameter file . . . . 73
`lm_pval` statements . . . . 74
`lm_twoqtl` introduction . . . . 89
`lm_twoqtl` sample output . . . . 97
LM-sampler . . . . 58, 63
location lod scores
  computational parameters . . . . 113
location lod scores computing requests . . . . 111
location lod scores estimates . . . . 89
location lod scores file
  identification statements . . . . 111
location lod scores mapping
  model parameters . . . . 112
location lod scores MCMC
  parameters and options . . . . 115
location lod scores output file description . . . . 112
location lod scores pedigree file description . . . . 112
location lod scores population
  model parameters . . . . 113
location lod scores statements . . . . 111
locus window . . . . 77
locus-by-locus sampling . . . . 61
locus-by-locus setup . . . . 59

## M

M-sampler . . . . 58, 61, 66, 92
major gene model . . . . 29
map function . . . . 33, 44, 48, 113
map test tlocs jointly at markers . . . . 98
marker allele frequencies . . . . 44
marker data . . . . 77, 119
marker data file . . . . 11, 65
marker names . . . . 44
marker seeds . . . . 45
marker simulation . . . . 43
marker simulation using meiosis indicators . . . . 43
marker simulation using trait . . . . 43
`markerdrop` computational parameters . . . . 45
`markerdrop` computing requests . . . . 43
`markerdrop` input file options . . . . 45
`markerdrop` introduction . . . . 37
`markerdrop` mapping model parameters . . . . 44
`markerdrop` output . . . . 41
`markerdrop` parameter file – conditional on
  inheritance pattern . . . . 40
`markerdrop` parameter file –
  conditional on trait . . . . 37
`markerdrop` population model parameters . . . . 44
`markerdrop` statements . . . . 43
Markov chain Monte Carlo . . . . 56, 63, 89

MC iterations ........ 60, 61, 115
MCMC introduction ........ 56
MCMC options ........ 59
MCMC parameter statements ........ 61
meiosis indicators .. 35, 37, 46, 54, 56, 61, 63, 89, 92
missing marker data ........ 77
missing quantitative trait data ........ 135
mixed model ........ 29
MORGAN ........ 1
MORGAN files ........ 9
MORGAN Gold standards ........ 5
MORGAN package structure ........ 5
MORGAN program libraries ........ 5
MORGAN subroutine libraries ........ 5
multiple meiosis sampler ........ 58, 63, 70, 89, 92
`multivar` ........ 135
`multivar` computational options ........ 141
`multivar` computational parameters ........ 140
`multivar` computing requests ........ 139
`multivar` output options ........ 141
`multivar` sample output ........ 137
`multivar` sample parameter file ........ 135
`multivar` segregation model parameters ........ 139
`multivar` statements ........ 139

## N

names; See individuals –names ........ 14
non-genetic model ........ 29

## O

observed individuals ........ 14, 140
output –redirect ........ 13
output control ........ 13
output warnings (W) ........ 13
overview of MORGAN ........ 1
overwrite file options ........ 12, 19

## P

parameter file ........ 10
parameter statement flexibility ........ 10
parameter statements ........ 10
parameter statements in pedigree files ........ 14
`pedcheck` examples ........ 19
`pedcheck` introduction ........ 18
`pedcheck` sample parameter file ........ 18
`pedcheck` statements ........ 20
pedigree component –See component ........ 18
pedigree file ........ 11, 14
pedigree file descriptions ........ 15
pedigree file separator ........ 14
pedigree options ........ 21
pedigree order ........ 21
pedigree peeling ........ 58
pedigree record format ........ 15, 45
pedigree size ........ 15
pedigree trait integer data order ........ 16
pedigree trait real data order ........ 16
pedigree validity ........ 18
penetrance ........ 45, 56
permutation based base-lod ........ 114
phased and unphased genotypes ........ 43, 58, 126
phased and unphased haplotypes ........ 132
phenotypic trait specification ........ 93
PolyEM ........ 135
PolyEM introduction ........ 135
PolyEM statements ........ 139
polygenic model ........ 29, 135
population kinship ........ 127, 134
population-based IBD inference ........ 117
population-based IBD inference
  parameter statements ........ 130
printlevel control ........ 13, 38
proband gametes ........ 48, 54, 66, 68, 77, 119, 134
pseudo-prior iterations ........ 60, 114

## Q

quantitative trait ........ 29, 78, 112, 135
quantitative trait specification ........ 94

## R

randomized p-values ........ 83
Rao-Blackwellized estimates ........ 96, 111
README documentation files ........ 5
recombination ........ 25
recombination breakpoints ........ 43, 54
rescue file ........ 75
residual variance ........ 31, 34
running `beaglesim` and `beagledag` ........ 121
running `fgl2ibd`, `fgl2haplo` and `fgl2dgl` ........ 126
running `genedrop` examples ........ 32
running `gl_auto` example ........ 71
running `ibddrop` example ........ 51
running `kin` example ........ 23
running `lm_auto` example ........ 67
running `lm_bayes` examples ........ 96
running `lm_linkage` examples ........ 95
running `lm_map` with genotypic data ........ 146
running `lm_map` with phenotypic data ........ 146
running `lm_pval` example ........ 73
running `lm_twoqtl` examples ........ 97
running `markerdrop` examples ........ 41
running `multivar` example ........ 137
running `simpop_fgl` ........ 123

**S**

sample by scan . . . . . 59, 61
sample by step . . . . . 59, 61
sample parameter file for `beaglesim` and `beagledag` . . . . . 120
sample parameter file for `civil` . . . . . 83
sample parameter file for `ibd_haplo` . . . . . 127
sample parameter file for `lm_bayes` . . . . . 91
sample parameter file for `lm_ibdtests` . . . . . 81
sample parameter file for `lm_linkage` . . . . . 91
sample parameter file for `lm_map` . . . . . 143
sample parameter file for `simpop_fgl` . . . . . 122
sample parameter files for `fgl2ibd`, `fgl2haplo` and `fgl2dgl` . . . . . 123
score file . . . . . 75
scoreset . . . . . 77, 134
seed file . . . . . 11, 30, 48
seeds for data simulation . . . . . 29, 32, 35, 36
seeds for sampler . . . . . 48, 55
selection against autozygosity . . . . . 50
selection against maternal identity . . . . . 50
sequential imputation . . . . . 59, 61
sequential imputation proposals . . . . . 116
simulating marker and trait data . . . . . 29, 35
simulation of a population . . . . . 122
simulation of descent in a pedigree . . . . . 46, 122
specifying inheritance . . . . . 56
state ordering: Jacquard . . . . . 133, 134
state ordering: lexicographic . . . . . 133

**T**

trait allele frequencies . . . . . 44
trait data . . . . . 45, 65, 78
trait model . . . . . 37
trait seeds . . . . . 45
trait test positions . . . . . 112
`translink` utility . . . . . 25

**U**

`unibig` . . . . . 135
`univar` . . . . . 135
unobserved individuals . . . . . 14, 29, 35
using the BEAGLE DAG . . . . . 120

**V**

vc-score and w-score . . . . . 80

**W**

whitespace . . . . . 9, 14
window patterns . . . . . 65, 77

# Statement Index

## A

allow component size ........ 12
allow observed individuals ........ 12
allow pedigree size ........ 12, 15
assign gender ........ 21
assume all observed ........ 36

## C

check eigenvalue computation ........ 141
check eigenvalues ........ 141
check ginverse ........ 141
check gmatrix ........ 141
check marker consistency [only] ........ 75
check progress MC iterations ........ 61
check trace ........ 142
compute [component] inbreeding coefficient ........ 25
compute [component] kinship coefficient ........ 25
compute [component] two-locus inbreeding coefficient ........ 25
compute eigenvalues ........ 141
compute estimates I times ........ 148
compute ibd statistics ........ 87
compute likelihood-ratio statistics ........ 87
compute scores every iterations ........ 61

## F

fit additive correlations ........ 140
fit additive covariances ........ 140
fit environmental model ........ 141
fit residual correlations ........ 140
fit residual covariances ........ 140

## I

ignore gender ........ 21
input eigenvalue file ........ 140
input extra file ........ 9, 11, 88
input gamete data file ........ 132
input genotypes as rows ........ 132
input haplotypes as columns of SNPs ........ 132
input haplotypes as rows of SNPs ........ 132
input individuals file ........ 11
input individuals record names traits reals ........ 112
input marker data file ........ 9, 11
input pedigree file ........ 9, 11
input pedigree record (father mother | father mother) ........ 16
input pedigree record gender (present | absent) ........ 16
input pedigree record inheritance integer pairs ........ 45
input pedigree record names 3 [integers] [reals] ........ 15
input pedigree record observed (absent | present) ........ 16, 36
input pedigree record trait integer pairs ........ 78
input pedigree record traits integers ........ 16, 45
input pedigree record traits reals ........ 16, 112
input pedigree size ........ 15
input rescue file ........ 75
input seed file ........ 9, 11

## L

limit breeding iterations ........ 140
limit EM iterations ........ 140
limit recombination fractions L ........ 147

## M

map [chromosome] marker positions base pairs ........ 132
map [chromosome] test tloc at markers ........ 113
map [chromosome] test tlocs jointly at markers ........ 113
map [chromosome][gender] marker [Kosambi] distances ........ 33
map [chromosome][gender] marker [Kosambi] positions ........ 33
map [chromosome][gender] marker recombination fractions ........ 33
map [chromosome][gender] test tloc all interval proportions ........ 112
map [chromosome][gender] test tloc beginning [Kosambi] distance ........ 113
map [chromosome][gender] test tloc beginning recombination fractions ........ 113
map [chromosome][gender] test tloc ending [Kosambi] distance ........ 113
map [chromosome][gender] test tloc ending recombination fractions ........ 113
map [chromosome][gender] test tloc external distance ........ 113
map [chromosome][gender] test tloc external Kosambi distance ........ 113
map [chromosome][gender] test tloc external recombination fractions ........ 113
map [chromosome][gender] test tloc intervals proportions ........ 112
map [chromosome][gender] tlocs marker [Kosambi] distances ........ 34

map [chromosome][gender] tlocs marker [Kosambi] positions 34
map [chromosome][gender] tlocs marker recombination fractions 34
map [gender] marker [Kosambi] distance 44
map [gender] marker [Kosambi] position 44
map [gender] marker recombination fraction 44
map [gender] tlocs marker [Kosambi] distance 44
map [gender] tlocs marker recombination fraction 44
map test tlocs no default external positions 113
map test tlocs no default interval proportions 113
map tlocs unlinked 33, 54, 75

## O

output [overwrite] extra file 12
output [overwrite] individuals file 12, 21
output [overwrite] pedigree file 11
output [overwrite] score file 12, 75
output [overwrite] seed file 12
output all genotypes 132
output final adjusted phenotypes 141
output final generation 132
output founder genome labels 76
output four-gamete state order jacquard 133, 134
output genotypes 131
output haplotype labels 131, 133
output haplotypes as columns 130
output haplotypes as rows 130
output maps gender [averaged] [specific] 147
output marker seeds only 36
output meiosis indicators 76
output pedigree chronological 21
output pedigree file 9
output pedigree record father mother 21
output pedigree record founder gene labels 35
output pedigree record mother father 21
output pedigree record trait latent variables 35
output pedigree record unobserved variables 35
output permutation seeds only 87
output Rao-Blackwellized estimates file 111
output rescue data iterations 76
output rescue file 75
output sampler seeds only 87
output scores at markers 134
output scores every scored MC iterations 76
output seed file 9
output spacing EM iterations 141
output statistics correlations 141
output statistics covariances 141
output trait seeds only 36
output with spaces 131, 133

## S

sample by scan 61
sample by step 61
select [chromosome] all markers 74
select [chromosome] all markers except 74
select [chromosome] markers 74
select inheritance 43
select partially phased data 133
select phased data 133
select state transition matrix 133
select trait 43, 74, 79, 139
select unphased data 133
set [chromosome] locus window 55, 77
set [chromosome] marker allele frequencies 34, 44
set [chromosome] marker names 44, 77
set [component] [scoreset] proband gametes 77
set [component] [scoreset] window patterns 77
set [component] proband gametes 54, 77
set [scoreset] proband gametes 134
set base log likelihood 113
set batches MC variance estimation 114
set breeding convergence 140
set burn-in iterations 61
set chromosome length centiMorgans 131
set debug level 14
set debug *libname* 14
set debug main 14
set dummy integers 13
set dummy reals 13
set eigenvalues 140
set EM convergence 147
set females per generation 131
set genotyping error rate 133, 148
set global MCMC 62
set ibd measures 87
set ibd permutations 87
set ibd tests 87
set inheritance tlocs 44
set kinship change rate 134
set L-sampler probability 61
set LD relaxation probability 130
set liability classes with penetrances 78
set likelihood-ratio lambda-p model gridpoints 87
set likelihood-ratio measures 87
set likelihood-ratio tests 87
set limit for exact computation 62
set LRT statistics iterations 148
set map estimation with [no] mistyping 148

set marker data ... 77
set marker seeds ... 35
set maximum ibdgraphs per component ... 113
set MC iterations ... 61, 115
set MCMC markers only ... 79
set million base pairs per centiMorgan ... 132
set normalized allele frequencies ... 34
set number of permutations ... 114
set offspring generations ... 131
set permutation seeds ... 87
set population kinship ... 134
set printlevel ... 13
set proband gametes all individual pairs ... 134
set pseudo-prior iterations ... 115
set pseudo-priors ... 114
set recombination frequencies ... 25
set remating probability weight ... 131
set SA ascent iterations ... 148
set SA convergence ... 148
set SA curvature iterations ... 148
set SA gradient iterations ... 148
set sampler seeds ... 55
set sequential imputation proposals
  every iterations ... 116
set standard deviations for genotypes ... 78
set test position window ... 116
set tlocs allele frequencies ... 34, 44
set trait data ... 78
set trait data discrete ... 45
set trait data discrete with covariate ... 78
set trait data discrete with liability ... 78
set trait effects ... 139
set trait for tlocs joint genotype means ... 114
set trait seeds ... 35
set traits additive variance ... 34
set traits for tlocs genotype means ... 34
set traits for tlocs
  incomplete penetrances ... 45, 78
set traits multiple tlocs ... 114
set traits residual variance ... 34
set traits tlocs ... 33, 43, 74
set transition matrix null fraction ... 133
set width of ages of onset window ... 79
simulate [chromosome] [dense] markers .. 43, 54
simulate [chromosome] markers ... 33
simulate ibd realizations ... 55, 114
simulate IBD realizations for trait ... 115
simulate tloc ... 54
simulate traits ... 33
start additive correlations ... 140
start additive covariances ... 140
start additive variances ... 139
start residual correlations ... 140
start residual covariances ... 140
start residual variances ... 139
start trait effect ... 139
start trait mean ... 139

## U

use exact summation for trait ... 114
use full EM ... 141
use locus-by-locus sampling for setup ... 61
use MCEM and SA for maximization ... 147
use Monte Carlo summation for trait ... 114
use multiplier realizations for null ... 115
use partitioned EM ... 141
use sequential imputation for setup ... 61
use sequential imputation
  realizations for setup ... 61