MORGAN Tutorial

Version 2.9

Last Updated: $Date: 2010-09-10 21:45:59 $

Table of Contents

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 (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 2.9 which was released in August 2008. It is available for download through the MORGAN V2.6 home page at the Department of Statistics, University of Washington.

The MORGAN programs are grouped into four categories:

1. Programs using deterministic algorithms: `pedcheck` checks for errors in pedigree structure and data format, see Chapter 3 [Checking Pedigree Validity], page 12. `kin` computes kinship and inbreeding coefficients for members of the pedigree, see Chapter 4 [Computing Kinship and One- or Two-Locus Inbreeding Coefficients], page 15.
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 19. `markerdrop` simulates marker data at loci linked to a potential trait locus, see Chapter 6 [Simulating Marker Data Conditional on Trait Data in Pedigrees], page 26. `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 33.
3. Programs using Markov chain Monte Carlo (MCMC) techniques: MORGAN's MCMC programs are split into two sections, "Autozyg" and "Lodscore". The Autozyg programs, `lm_auto` and `lm_pval`, estimate conditional gene *ibd* probabilities, see Chapter 9 [Estimating Conditional IBD Probabilities by MCMC], page 43. The Lodscore programs, `lm_lods`, `lm_markers`, `lm_bayes` and `lm_schnell` estimate multilocus LOD scores, see Chapter 11 [Estimating Location LOD Scores by MCMC], page 58. A brief introduction to the MCMC techniques employed by MORGAN can be found in Chapter 8 [Using MCMC to Estimate Parameters of Interest in Pedigree Data], page 38.
4. Programs using EM algorithm for segregation analysis with quantitative traits: includes `univar`, `unibig`, `bivar` and `multivar`, see Chapter 12 [Polygenic Modeling of Quantitative Traits by EM Algorithm], page 71.

This tutorial is based on the computing notes of Dr. Elizabeth Thompson. Descriptions of the parameter statements are taken from the '`README_userdoc`' files written by Myrna Jewett. The original version of this document was written in 2002 by Michael Na Li, and revised by Myrna Jewett and Adele Mitchell in 2006. It was revised again for MORGAN 2.8.3 with new Examples by Audrey Q. Fu. The current version together together with updated Examples for MORGAN 2.9 was by Tia Lerud in 2009, further revised by Marshall Brown in 2010.

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.

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 are also available to download for off-line reading or printing.

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. Unpack the examples by typing the following command in a shell window,

   ```
   user$ tar zxvf morgan-examples_V29.tar.gz
   ```

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

   ```
   user$ gunzip -c morgan-examples_V29.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.)

2. 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 `vi` or `pico`) 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 = /castor/genepi/MORGAN_V2.9
   EXAMPLEDIR = /h4/audrey/MORGAN_Examples
   BINDIR = /h4/audrey/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
   #          BINDIR is your bin directory
   ```

```
# BINDIR is needed only if you prefer to link to executables from your
#    bin directory, rather than running from a current directory: for
#    example, if your current directory is not in your standard path.
```

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 .

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 30-individual 5-generation JV pedigree derives from a real study of a rare recessive trait by Goddard et al (1996 AJHG 58: 1286-1302).

The other pedigree in '`ped73.ped`', located under '`MORGAN_Examples`', consists of three components and 73 individuals: component one has 47 individuals from 6 generations, component two 11 individualus from 3 generations, and component three 15 individuals from 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 under '`PedInfo`'.

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 documentaion, Makefiles, and examples. These are therefore described in this section, updated for the released version of MORGAN 2.9.

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 V2.9 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. For example, in the first release of MORGAN 2.9 there is a README_bugs_v2.9: these bugs are now fixed.

2. README files in main program directories

   The main program dirctories of MORGAN 2.9 are PedComp, Genedrop, Autozyg, Lodscore, LR_Lods, 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.

   Occasionally, there may be additional README files. For example the Autozyg main program directory contains a file README_IMPORTANT, described the (few) parameter statements that must be changed when moving from older to newer versions of MORGAN.

   Most subroutine directories do not contain README files. An exception is the library of subroutines for the lm_twoqtl program, '`/TwoQTL`'. The README_twoqtl file in the '`TwoQTL`' subdirectory contains information about running lm_twoqtl. This information is not yet incorporated into the relevant main-program directory README_userdoc file.

   Additionally, the Autozyg main program directory contains a subdirectory '`/Utils`' of routines specific to the programs lm_ibdtests and lm_map which have not yet been incorporated into a library directory. There are two README files in '`Utils`': README_ibdtests and README_lm_map written by the authors of those programs. In this case, most of the relevant information is now incorporated into the README_userdoc in the Autozyg program directory.

3. README files in Gold and Test subdirectories.

   Each main program directory contains a subdirectory, Gold, or in two cases both a Gold and a Gold1 subdirectory. 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 MORGAN_EXAMPLES files used in the tutorial. Each Gold and Gold1 subdirectory contains a README_gold file detailing the examples in that directory.

   Additionally, in some versions of MORGAN, Autozyg contains a TestGL subdirectory, with an example for testing installation of the real-time graphics display (if installed). The README_display file in this subdirectory provides information on this topic.

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 libraries can be divided broadly into four groups:

   - Lowest-level libraries required by all programs
     - Stuff: Routines for printing, allocating, freeing
     - Pars: Routines for processing MORGAN parameter statements
   - Low-levels libaries 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
   - Main libraries supporting genetic analysis programs
     - Pedchk: Routines for checking validity of input pedigrees. Routines for the 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 analysing 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.
   - Extra specialist libraries
     - GLDisp: Routines for the real-time graphics display
     - TwoQTL: Routines for the Lodscore program lm_twoqtl

     In addition to the subroutine libraries, the subdirectory '`Utils`' of Autozyg contains code for subroutines that are directly incorporated into the lm_ibdtests and lm_map 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. They may eventually be incorporated into the MORGAN subroutine libraries.

     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 main program directories

  The main program directories of MORGAN 2.9 are PedComp, Genedrop, Autozyg, Lodscore, LR_Lods, and PolyEM. When MORGAN is installed these directories contain the following executables:

  - PedComp: pedcheck, kin
  - Genedrop: genedrop, ibddrop, markerdrop
  - Autozyg: lm_auto, lm_pval, lm_ibdtests, lm_map
  - LR_Lods: lm_lods, lm_schnell
  - Lodscore: lm_bayes, lm_markers, lm_multiple, lm_twoqtl
  - 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.

  To make room for new Lodscore programs, with MORGAN V2.8.2, the new directory LR_Lods was created to contain the two older programs `lm_schnell` and `lm_lods`. These two programs differ in several ways from newer programs, but the principal one is that they use the methods of combining likelihood ratios (LR) along the chromosome in order to estimate lod scores (see Thompson & Guo, 1991, IMA J Math Appl in Med & Biol).

- GOLD-STANDARD directories, Makefiles and examples

  The Gold and Gold1 subdirectories of the main program directories PedComp, Genedrop, PolyEM, Autozyg, LR_Lods and Lodscore contain example runs of all the main programs in order to test various aspects of code and installation. Examples for a paticular main program are in the Gold and/or Gold1 subdirectory of that main program directory.

  In MORGAN V2.8.3, `Gold` replaces the previous `Gold2` subdirectory, in Autozyg, LR_Lods, and Lodscore directories. `Gold1` directories remain in Autozyg and LR_Lods directories since they provides the only tests of MCMC samplers on looped pedigrees. `Gold1` gold standards were omitted from the released MORGAN V2.8.2, due to delays in checking looped pedigree peeling routines, but were reinstated from MORGAN V2.8.3.

  The Gold and Gold1 subdirectories typically contain numerous test parameter files, pedigee 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 or Gold1 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.

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).

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_lods`. 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
`mark`	Input marker data file (Note that only Autozyg programs use marker data)
`oped`	Output pedigree file
`seed`	Input seeds for random number generator
`oseed`	Output random seeds
`oscor`	Output score file

If the name for a particular file type 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.

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.

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`’.

```
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 pedigree file ‘check.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.

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.

`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.

`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 9.6.7 [Autozyg computational parameters], page 51.

`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] 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.

2.4 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 some MORGAN users find it convenient to describe the pedigree data within the 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. 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.

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, genotype data are not included in a MORGAN pedigree file.

2.5 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 12 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.

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 6 reals 1
```

```
***************************************************
101 0 0 1 0 0 0 -1 -1 999.5
102 0 0 2 0 0 0 -1 -1 999.5
201 101 102 1 0 0 0 0 1 999.5
```

Note that genotype data are not contained in the pedigree file; genotype 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 9 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 all individuals are assumed to be observed. The 8 integers following gender and the real number in the final column represent trait data. Lack of an '`input pedigree record traits integers`' statement indicates that the first integer following gender corresponds to trait 1, the second to trait 2, etc. 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.

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

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
– 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 name), 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.

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`:

```
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 pedigree file 'check.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.

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`'. You will get an error message and the program will quit if '`check.oped`' already exists. If this occurs, delete the file and try again or use another output file name.

`./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, with input pedigree file '`sex.ped`' specified on the command line. 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. What does the output say is wrong with this pedigree?

3.4 `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 optional seed file, see Section 2.3 [File identification statements], page 8.

- For describing the format of the pedigree file, see Section 2.5 [Pedigree file description statements], page 10.

`(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.

4 Computing Kinship and One- or Two-Locus Inbreeding Coefficients

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: currently the `kin` program does not check for duplicate requests within a pedigree component of any inbreeding or kinship coefficients. It does check (and quits with an error) if a request is made for kinship of an individual with him/her self. These bugs will be fixed in a future MORGAN release.

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`'.

```
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 2 – 7 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.

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
```

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 squared one-locus inbreeding coefficient.

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 recom-

bination 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.

5 Simulating Marker and Trait Data in Pedigrees

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 genes inherited and the founder labels. 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, a *founder gene label* or *founder 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 be diallelic 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 2.4 [Pedigree file], page 9, for details.

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`'. *SEED FILE:* 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 24.

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:

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

The first line specifies '`marker.seed`' as the input seed file for the marker simulation. The default behavior is to save both the marker and trait seeds. The second statement, '`output marker seeds only`', overrides this default behavior and so 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, hence the second and third lines in the above example are commented out in '`ped73_gdrop.par`'. See the next section for command line implementation.

Other notes on the parameter file: The statement '`output pedigree chronological`' is included in the example '`.par`' files so that the output pedigree will be in the chronological order required for use with other MORGAN programs.

```
simulate chrom 1 markers 10 traits 1
```

The above statement asks `genedrop` to simulate ten markers loci and one trait locus on chromosome 1. If no trait locus is to be simulated, the part '`traits 1`' can be removed.

```
map chrom 1 marker dist  10 10 10 10 10 10 10 10 10
```

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.

A marker map 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 freqs 0.13 0.66 0.16 0.05
set chrom 1 markers 2   freqs 0.06 0.23 0.41 0.25 0.05
set chrom 1 markers 3   freqs 0.11 0.02 0.01 0.06 0.24 0.56
set chrom 1 markers 4   freqs 0.07 0.04 0.89
set chrom 1 markers 5 freqs 0.12 0.11 0.03 0.03 0.50 0.21
set chrom 1 markers 6 freqs 0.50 0.44 0.06
set chrom 1 markers 7 freqs 0.01 0.33 0.62 0.04
set chrom 1 markers 8 freqs 0.20 0.05 0.42 0.27 0.06
set chrom 1 markers 9 freqs 0.18 0.18 0.25 0.16 0.08 0.15
set chrom 1 markers 10 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 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 locus. The trait locus is between markers 5 and 6 on chromosome 1, at a distance of 5 cM to marker 5. 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.

```
     map chrom 1 trait 1 marker 5 dist 5
     set trait 1 freqs 0.5 0.5

     set trait 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 gene labels
     output pedigree record trait latent variables
     output pedigree record unobserved variables
```

These lines request that the founder gene (or genome) labels and latent variable values for the trait be included in the output file, and that the data be simulated for all (observed and unobserved) individuals. Founder gene labels indicated, for all non-founders, which founder alleles were passed to the individual. The latent and the additive and residual components of the trait value. Latent trait variables will precede the trait value in the output file.

5.3 Running genedrop examples and sample output

Two examples are available under the subdirectory '`Simulation/`'. The commands to run these examples are similar to the following (see the '`README`' file in the '`Simulation`'directory for additional options):

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

When running the `genedrop` example, notice (but do not worry about) the appearance of a warning message on the screen: '`Overrides previous statement of same type (W)`'. The warning was triggered by the specification of '`seeds`' as both the input and output seed file, without inclusion of '`overwrite`' in the '`output seed file`' statement.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. To avoid this warning (and an ever-growing seed file), one can access the seed file from the parameter file and use the '`overwrite`' option when outputting the seeds (see the previous section Section 5.2 [Sample genedrop parameter file], page 19).

Since the function of `genedrop` is to simulate marker and trait data, it, unlike other MORGAN programs, always creates and output 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 gene labels (FGLs) for markers precede the marker genotypes and the trait FGLs precede the trait locus genotypes (only if latent variables are requested, as in this example).

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.

5.4 `genedrop` statements

5.4.1 `genedrop` computing requests

`simulate [chromosome` *I*`] markers` *J* `[traits` *K*`]`

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 given. The '`chromosome`' keyword can be omitted if all markers and linked traits are on the same chromosome. The markers and traits are labeled as positive integers. The marker labels are *relative* to each chromosome whereas the trait labels are *absolute*. That is, the markers on chromosome 5 are labeled 1, 2, . . . , and so are the markers on chromosome 7. The traits however are labeled, for example, as 1 on chromosome 5 and as 2 on chromosome 7. There can be at most 10 traits and 500 markers.

`simulate unlinked traits` *K1*`...`

Optional. This statement specifies traits to be simulated which are not linked to markers, and hence, have no map specifications.

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 must be provided such that the number of distances is one less than the number of markers. If '`positions`' is chosen, the number of positions must 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)] traits` *`K1 K2`* `... markers` *`J1 J2`* `... ( [Kosambi] distances | recombination fractions)` *`X1 X2...`*

This statement is required if simulated traits are to be linked to markers. That is, it is not required if no traits or only unlinked traits 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 traits 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 trait 3 2 marker 6 0 distances 5 4`', traits 3 and 2 will be placed 5 cM to the right of marker 6 and 4 cM to the left of marker 1, respectively.

5.4.3 `genedrop` population model parameters

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

This statement specifies the trait allele frequencies. Allele frequencies for a trait 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 be biallelic.

`set normalized 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`* `... 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.

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.

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 gene 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 labels are determined by the simulated meiosis indicators.

  This statement is useful in cases where we want the “disease” allele in the pedigree to come from a particular founder.

`output pedigree record trait latent variables`
: This statement requests that the quantitative trait value be included in the output for each trait. 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 gene 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.

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.

6 Simulating Marker Data Conditional on Trait Data in Pedigrees

6.1 Introduction to `markerdrop`

`markerdrop` simulates marker data at markers linked to a potential trait locus. The user must specify whether marker data simulation is to be conditional on a trait model or on an inheritance pattern at the trait locus. 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.

- If marker data simulation is to be conditional on a trait model, parameters must be provided for trait locus allele frequencies using '`set traits frequencies`', for genotypic penetrances using '`set incomplete penetrances`', and for the map position of the trait locus using a '`map`' statement see Section 6.5 [markerdrop statements], page 30. There must be only one mapping statement for the trait; from this statement the trait 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 inheritance indicators. For more information on inheritance indicators, see Section 6.5.1 [markerdrop computing requests], page 30 and Chapter 8 [Using MCMC to Estimate Parameters of Interest in Pedigree Data], page 38. Location of inheritance 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.

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. The relevant section of the file is:

```
map trait 2 marker 5 distance 5.0
simulate markers 10 using trait
map marker positions 10 20 30 40 50 60 70 80 90 100
set incomplete penetrances .05 .6 .95
set trait 2  freqs 0.5 0.5

set markers 1 freqs 0.13 0.66 0.16 0.05
set markers 2   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 three lines are required for `markerdrop` and must be included in the parameter file. The '`map trait`' 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 10 using trait`' statement indicates the number of markers to be simulated and specifies that the markers will be conditional on a trait model. The '`map marker positions`' statement specifies the spacing of the markers to be simulated.

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 (lines 4 and 5) are required for discrete traits (the default for simulation on a trait). Line 4: '`set 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. Line 5: '`set trait ... freqs`' specifies trait allele frequencies.

The '`set markers`' statements beginning at line 6 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 translates into a specification of which alleles are to be used in simulating marker data.

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 6 reals 1

*************************************************
101 0 0 1 0 0 0 -1 -1 999.5
102 0 0 2 0 0 0 -1 -1 999.5
201 101 102 1 0 0 0 0 1 999.5
202 101 102 2 0 0 0 1 1 999.5
2010 0 0 2 0 0 0 -1 -1 999.5
```

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 affection status for two possible discrete traits (0=missing, 1=unaffected, 2=affected). The affection status is to be used is specified through statements 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.

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.

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:

```
map trait 3 marker 4 recomb frac 0.01
simulate markers 10 using inheritance
map marker positions 10 20 30 40 50 60 70 80 90 100

set markers 1 freqs 0.13 0.66 0.16 0.05
set markers 2   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 three lines are required and must be included in the parameter file. The '`map trait`' 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 4 with a recombination fraction of 0.01. The '`simulate markers 10 using inheritance`' statement indicates the number of markers to be simulated and specifies that the markers will be conditional on the inheritance pattern. The '`map marker positions`' statement specifies the spacing of the markers to be simulated. 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 translates into a specification of which alleles are to be used in simulating the 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 setion of this file are below.

```
input pedigree size 73
input pedigree record names 3 integers 4

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

The first three columns are indices of individuals and their parents. The next two are sex and observation status. The last two integer columns are inheritance indicators with the first being the paternal ones and the second the maternal ones. A founder’s inheritance 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 indeed the last two integer columns in this pedigree file.

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 30.

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. A section of this output file is given below. Note that similar output would be generated using '`ped73_mdrop_trait.par`'.

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

                           T3
----------------------------+---------------------------------------
  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 (FGLs) 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 FGLs. One of each founder's FGLs 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.

6.5 `markerdrop` statements

6.5.1 `markerdrop` computing requests

`markerdrop` requires one of the following two statements:

`simulate markers` *J* `using trait`

This statement requests the simulation of *J* markers conditional on a trait model. 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.

`simulate markers` *J* `using inheritance`

This statement requests the simulation of *J* markers conditional on an inheritance pattern at the trait locus. If marker data are to be simulated conditional on an inheritance pattern, the user must specify a map position for the trait within the parameter file. In addition, a pair of inheritance 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.

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)] trait` *K* `marker` *J* `( [Kosambi] distance | recombination fraction )` *X*

This statement is required for `markerdrop`; it tells the program which trait 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`'.

6.5.3 markerdrop population model parameters

`set trait` *K1* `frequencies` *X1 X2*

This statement specifies trait allele frequencies. Trait must be biallelic; 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 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 markers` *`J1 ...`* `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.

6.5.4 `markerdrop` computational parameters

`set incomplete penetrances`

This statement is required for `markerdrop` when using a trait model or when using inheritance 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 ‘`set 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 data discrete`

This statement is optional. A discrete trait is the default when simulating using a trait model.

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 24.

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 traits` *`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 inheritance 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.

7 Estimating *a priori ibd* Probabilities by Monte Carlo

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.

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 haplotypes can be scored, or it can be scored jointly over a moving window of several loci. If the moving window option is selected, `genedrop` calculates the probability that the specified pair of gametes are *ibd* at all loci in the window. As a result, it is then possible to determine the probability that all or some of the gametes are *ibd* for a particular haplotype.

7.2 Sample `ibddrop` parameter file

Files for `ibddrop` may be found in the '`IBD`' subdirectory of '`MORGAN_Examples`'. The sample parameter file for `ibddrop` is '`jv_rep_ibd.par`'.

```
input pedigree file "jv_rep.ped"

simulate markers 5  trait 1

map           markers   distances 44.6 44.6 11.2 11.2
map trait 1  marker 2  distances 22.3

set component 1  proband gametes 331 0 333 1
set component 2  proband gametes 541 0 541 1 341 0 343 1

input seed file '../sampler.seed'

set MC iterations 20000
```

The parameter file specifies the pedigree file name '`jv_rep.ped`' and then asks for five markers and one trait locus. Since there are no data, the distinction between marker and trait doesn't mean anything – it is just a way to specify a set of loci, one of which may be unlinked. '`jv_rep.ped`' contains data on 30 individuals, including gender and one trait.

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 36).

The number of Monte Carlo iterations is set to be 20,000 by the '`set MC iterations`' statement.

Note that if one would like to compute multilocus ibd probability, the statement '`set locus window`' can be used to specify number of loci to score simultaneously. `ibddrop` has limited functionality for computing multilocus probabilities, it can only examine two gametes to determine whether or not the two are *ibd*. For instructions on how to implement windows in this example, see the parameter file. For additional options, including specific patterns over two or more gametes, see Section 9.2 [Sample lm_auto parameter file], page 43.

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.

The '`ibddrop`' example can be run under the subdirectory '`IBD/`' with the following command:

```
./ibddrop jv_rep_ibd.par > ibddrop.out
```

The genetic map specified by the statements '`map markers distances`' and '`map trait 1 marker 2 distances`' is below. Note the position of the trait locus (*T1*) with respect to the marker loci.

```
Distances (cM):

             T1
--------------+---------------------
  44.6   22.3   22.3   11.2   11.2
+------+-------------+------+-------+
    M1      M2              M3      M4      M5
```

Since the parameter file contains two '`set proband gametes`' statements, `ibddrop` will produce two 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:  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    .0322    .0312    .0289    .0301    .0294    .0306        0
      1 1 1 2    .0291    .0294    .0295    .0284    .0299    .0295        1
      1 1 2 1    .0129    .0143    .0147    .0140    .0142    .0134        3
      1 1 2 2    .0103    .0098    .0097    .0097    .0089    .0092        4
      1 1 2 3    .0273    .0280    .0263    .0269    .0276    .0267        5
      1 2 1 1    .0654    .0638    .0634    .0652    .0633    .0649        6
      1 2 1 2    .0060    .0063    .0057    .0062    .0062    .0058        7
      1 2 1 3    .0571    .0576    .0581    .0601    .0602    .0584        8
      1 2 2 1    .0692    .0680    .0661    .0685    .0703    .0728        9
      1 2 2 2    .0483    .0527    .0488    .0493    .0486    .0485       10
      1 2 2 3    .1389    .1405    .1386    .1384    .1371    .1326       11
      1 2 3 1    .1348    .1366    .1354    .1379    .1369    .1391       12
      1 2 3 2    .0267    .0250    .0259    .0266    .0267    .0285       13
      1 2 3 3    .0949    .0945    .0980    .0949    .0968    .0975       14
      1 2 3 4    .2469    .2424    .2508    .2437    .2442    .2427       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.

To compute multilocus ibd probabilities, say for 3 loci, follow the instructions to use '`set locus window 3`' in the parameter file and re-run the example using the same command line. The interesting part of the output is:

```
Summary for component 2:

   Probabilities of IBD patterns for windows of 3 loci

      Proband gamete set 1:  541 0  541 1

        IBD  wndw 1 wndw 2 wndw 3 wndw 4
```

```
0 0 0   .7291  .7443  .7657  .7881
0 0 1   .0698  .0655  .0482  .0478
0 1 0   .0640  .0532  .0365  .0266
0 1 1   .0279  .0252  .0369  .0284
1 0 0   .0806  .0696  .0703  .0493
1 0 1   .0087  .0080  .0067  .0049
1 1 0   .0135  .0238  .0177  .0268
1 1 1   .0063  .0105  .0180  .0281
```

This time, `ibddrop` was asked to compute *ibd* probabilities in windows of three loci at a time. This was done using the '`set locus window`' statement. Since the trait locus is unlinked to the marker loci in this example, it is placed to the left of the five marker loci on the map. Thus the first window, '`wndw 1`' in the table above, includes the trait locus and the first two marker loci, '`wndw 2`' includes the first three marker loci, '`wndw 3`' includes marker loci 2, 3 and 4, etc. 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 541 are *ibd* at marker loci 3 and 5, but not at marker locus 4 is 0.0049.

Note that there are two additional example parameter files in the '`IBD/`' subdirectory; these examples are not discussed in the tutorial but are there for the interested user.

7.4 `ibddrop` statements

- Use the '`simulate`' statement 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 22).
- Use '`map`' statements to specify the marker and trait maps (see Section 5.4.2 [genedrop mapping model parameters], page 22).

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 46), where simulation is conditional on marker and trait data.

`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.

`set MC iterations` *`I`*

Required. This statement specifies the total number of Monte Carlo iterations.

`simulate markers` *`K1`* `trait 1`

This statement specifies the number of markers to be simulated, as well as a linked trait. A linked or unlinked trait must be specified.

`simulate unlinked trait 1`

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.

8 Using MCMC to Estimate Parameters of Interest in Pedigree Data

8.1 Specifying inheritance

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*. 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 called *inheritance indicators*. 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 inheritance indicators is notated as **S** = (*Sij*) where

Sij = 0 if DNA involved in meiosis *i* at locus *j* is the gamete's parent's maternal DNA
1 if DNA involved in meiosis *i* at locus *j* is the gamete's parent's paternal DNA

S.j is referred to as the *inheritance vector* at locus *j*. It is the list of the inheritance indicators for all meioses in the sample at a single locus (locus *j*). The elements of *S.j* are independent of one another, as they represent the grandmaternal or grandpaternal inheritance for each gamete at the locus and meiosis events are independent of one another. *Si.* is the list of the inheritance indicators at all loci for a single meiosis (i.e., within a single gamete). The elements of *Si.* have first–order Markov dependence. That is, the probability of a '`0`' or '`1`' at locus $j + 1$ is a function of the the value at locus j and the recombination fraction between the loci. In other words, the value at locus $j + 1$ is conditionally independent of all other loci, given the value at locus *j*.

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

8.2 Genetic model

In earlier versions of MORGAN there were three genetic data types. Data could be *genotypic*, typically used for marker data, *discrete*, a data type for binary data requiring specification of *incomplete penetrances*, or *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 are diallelic. 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 (1976: University of Utah, Bioinformatics

Tech Rept, #6). For zero-loop pedigrees, 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:

   From MORGAN V2.8.2, liability classes (previously available only for `lm_bayes`) have been implemented for the discrete-trait penetrance model in `lm_markers` and `lm_multiple`. Penetrances for each liability class are now read from an input file using the "input extra data file S" parameter statement.

   Additionally, an age-based penetrance function for a qualititative trait has been implemented. That is, penetrances are directly dependent on age, rather than going through a liability class specification.

3. Traits and trait loci:

   The new program `lm_twoqtl` allows two (linked or unlinked) quantitative trait loci to contribute additively or epistatistically to a single trait (see Sung et al., 2007, *Genetic Epidemiology* **31:** 103-114). A polygenic component may also be also be included. In 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 (see Sung & Wijsman, 2007, *Human Heredity* **63:** 144-152.).

   With more these more complex trait models, including those of `lm_twoqtl` (see Sung et al., 2007, *Genetic Epidemiology* **31:** 103-114), a more general specification of traits is required. In 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.

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.

From MORGAN 2.8, exact computation is performed on small pedigree components. Further, these HMM computations are also a component of MCMC sampling on larger pedigree components (see next section).

Note that in fact Sij are independent over meioses i, so that the structure is that of a factored HMM. In MORGAN V2.8.2, 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_markers` and `lm_multiple`: computation uses the FHMM version of the Baum algorithm.

In MORGAN V2.8.2, Gold standards for exact computation are added in the Lodscore/Gold2 subdirectory.

8.4 LM-sampler and LMM-sampler

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 inheritance 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 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 inheritance indicators at that locus are updated based on the genotype data at all loci and on the current realization of inheritance 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. This method is a modification of the Elston-Stewart algorithm (Elston RC and Stewart J (1971) *Human Heredity* **21**:523-542) and it 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 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 inheritance indicators for that meiosis are updated conditional on the genotype data at all loci and the current realization of inheritance indicators for all other meioses. The M-sampler is a modification of the Lander-Green algorithm (Lander ES and Green P (1987) PNAS 84:2363-2367) for peeling along a chromosome using the Baum algorithm (Baum LE (1972) *in* O. Shea, ed., 'Inequalities-III; Proceedings of the Third Symposium on Inequalities, UCLA, 1969', Academic Press, NY pp. 1-8). 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.

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 mathematical details on the L-, M- and LM-samplers, see Thompson EA (2000) *Statistical Inference from Genetic Data on Pedigrees*, *in* 'NSF-CBMS Regional Conference Series in Probability and Statistics, Volume 6. Institute of Mathematical Sciences, Beachwood OH and American Statistical Association, Alexandria VA.

Multiple meiosis (MM) sampler updates multiple meioses jointly and is therefore a generalization of the meiosis sampler (M-sampler). There are four types of update in MM-sampler: random meiosis update, individual update, sib update and 3-generation update. This is based on work by Liping Tong (Tong L and Thompson EA (2008) *Human Heredity* **65:** 152-163). The LMM-sampler is a combination of L-sampler and MM-sampler, analogous to the LM-sampler. The LMM-sampler is implemented in the program `lm_multiple`, and from MORGAN 2.8.3, `lm_markers` is compiled as a special case of `lm_multiple`. The LMM-sampler can also be used in the program `gl_auto`, a MORGAN 3.0 program to sample inheritance patterns conditional on marker data.

From MORGAN V2.8.2, Gold standards for `lm_multiple` are added in the Lodscore/Gold2 subdirectory.

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.

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 V2.8.3, and specifically to accommodate the new `lm_haplotype` program, 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. In this case, the “set peeling by component” option is retained.

8.5 MCMC parameters and 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 contruct 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 L-sampler 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 meiosis (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 the desired stationary distibution. 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. **Recommended number of iterations is on the order of 10^5**. `lm_bayes` performs a third type of iteration to calculate pseudo-priors. Alternatively, pseudo-priors can be read from an input file. They 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 two 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 inheritance 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 this 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 inheritance indicators that are recombinant and non-recombinant in each interval of the map.

In these diagnostic scores, for the programs `lm_pval`, `lm_markers` and `lm_multiple` 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. For programs `lm_schnell` and `lm_lods` the trait locus (designated '0') is included in its position in that cycle of MCMC. If poor MCMC mixing is suspected, it can be useful to see if these diagnostic probabilities and counts differ significantly among MCMC runs.

9 Estimating Conditional *ibd* Probabilities by MCMC

9.1 Introduction to `lm_auto` and `lm_pval`

The MORGAN programs `lm_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 perform multipoint linkage analysis 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 and affectation status (affected / unaffected / unknown) for the trait of interest. `lm_auto` and `lm_pval` estimate conditional probabilities of gene *ibd* states, given the trait and marker data.

`lm_auto` uses the LM-sampler to realize *ibd* configurations from their conditional distribution given the marker data. Given marker data, it estimates conditional probabilities of genome sharing patterns (gene *ibd*) among specified haplotypes, usually from affected individuals. The marker data are used jointly in the sampling, but the resulting *ibd* is scored marginally at each marker locus.

`lm_pval` also uses LM-sampling to provide the conditional distribution of an *ibd* measure 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 either as a list of names in the parameter file or as the phenotypic status in the pedigree file.

The version of the program `lm_pval` released in MORGAN V.2.8 and subsequent, and described in this tutorial, uses the latent p-value distribution of Thompson & Geyer (2007, Biometrika). In `lm_pval`, marker data are assumed available on some pedigree members, at some of the marker loci. The distribution of the *ibd* measure conditional on marker data is compared to the unconditional distribution under the null hypothesis of no linkage to produce quantiles of a latent (fuzzy) p-value distribution. A latent p-value distribution corrected for multiple testing is also produced, by scoring the maximum of the *ibd* measure over loci.

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`). These are working names only; versions of the programs will be released under MORGAN 3. The methods are described in Thompson (2008: JSM 2007 proceedings, Pp. 3751-3758). The MORGAN 3 program `civil` also uses latent p-values (Di and Thompson, 2009, Human Heredity 68: 139-150).

9.2 Sample `lm_auto` parameter file

`lm_auto` uses the parameter file '`jv_rep_auto.par`':

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

select all markers  traits 1

map gender F  markers           distances 25.5 25.5 25.5 25.5
map gender M  markers            distances 11.2 45.8 11.2 45.8
map gender F  trait 1  marker 2  distances 12.8
map gender M  trait 1  marker 2  distances 5.8

set markers 1 2 3 4    freqs  .2 .2 .4 .1 .06 .04
set markers 5          freqs  .3 .2 .3 .1
set trait   1          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
set MC iterations 2000
```

The trait values are specified in the parameter file and 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. 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 names and gender in the pedigree file. The specified pedigree file, '`jv_rep.ped`', is a 30-member, two-component pedigree in which the final individuals (named 531 and 541) have trait value '`4`'. All other individuals in the file have trait value '`0`'. Because the trait type is not specified in the parameter file via a '`set trait data`' statement, the trait type is assumed to be *genotypic*. This means that the trait locus genotype can be inferred from the trait value, i.e. there are three distinct trait values, each corresponding to a distinct genotype at the trait locus.

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 22. 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 '`select`' statement is analogous to `genedrop`'s '`simulate`' statement (see Section 5.4.1 [genedrop computing requests], page 22).

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.

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*) accross several loci.

Recall that in `ibddrop`, one can compute the probability of two gametes being *ibd* or not. The values in the '`IBD`' column of the output indicate whether the 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 *ibd* patterns of interest over two or more 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 34. 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 run `ibddrop` before using `lm_auto` to compute 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 as of MORGAN V2.9, the number of proband gametes is limited to 10. See Section 7.4 [ibddrop statements], page 36, 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).

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 first table of calculations (about halfway through the output) gives probabilities of gene *ibd* patterns for each marker and the trait locus (in the map order).

```
          Probabilities of IBD patterns

             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    .1985    .2960    .3480    .2920    .2430    .1840        0
             1 1 1 2    .1400    .2280    .2690    .2205    .1720    .1145        1
             1 1 2 1    .1265    .1495    .2050    .1225    .0975    .0985        3
             1 1 2 2    .0145    .0165    .0200    .0145    .0125    .0070        4
             1 1 2 3    .0340    .0350    .0515    .0255    .0220    .0130        5
             1 2 1 1    .0235    .0150    .0030    .0095    .0185    .0275        6
             1 2 1 2    .0755    .0425    .0110    .0640    .0705    .0775        7
             1 2 1 3    .0595    .0330    .0085    .0310    .0515    .0570        8
```

```
    1 2 2 1    .0380    .0260    .0030    .0275    .0300    .0435        9
    1 2 2 2    .1035    .0605    .0255    .0860    .1175    .1530       10
    1 2 2 3    .0540    .0370    .0150    .0385    .0490    .0580       11
    1 2 3 1    .0200    .0080    .0035    .0060    .0140    .0210       12
    1 2 3 2    .0495    .0255    .0155    .0385    .0645    .0700       13
    1 2 3 3    .0100    .0045    .0060    .0030    .0095    .0170       14
    1 2 3 4    .0530    .0230    .0155    .0210    .0280    .0585       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   .5325  .4900  .4740  .5385
    0 0 1   .0975  .0870  .0625  .0685
    0 1 0   .0295  .0440  .0400  .0550
    0 1 1   .1275  .0665  .0555  .0315
    1 0 0   .0445  .0465  .1330  .1125
    1 0 1   .0130  .0085  .0240  .0250
    1 1 0   .0255  .1130  .0975  .1030
    1 1 1   .1300  .1445  .1135  .0660
```

Interpretation of these results is similar to that of `ibddrop` See Section 7.3 [Running ibddrop example and sample output], page 34. 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 34. 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 above. 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 parameters and options], page 41.

9.4 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'

set component 1 affected individuals 302 307 406 407 408 411 414 416 505 507 508 511 5
set component 2 affected individuals 1301 1302 1304
set component 3 affected individuals 2301 2302 2306 2307 2308 2305

#input pedigree record trait 1 integer 4
#set affected individuals 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 trait marker`' statements are included and the trait locus is not included in the '`select`' statement. 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`. Affected individuals may be specified explicitly or determined by using trait data. In the above example, the three '`set component []`

`affected individuals`' statements explicitly name each affected individual for each component. Below these three statements are two commented out statements, which can be used instead of the '`set component [] affected individuals`' statements to specify affected individuals based on trait data. The first of these statements, '`set affected individuals trait 1`', instructs the program to determine the affected individuals by using the trait data for trait 1 in the pedigree file. Recall that the second commented out 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. See the parameter file for further instruction on how to implement the two methods for specifying the affected individuals.

9.5 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     0.001     0.001
               0.000     0.000     0.000
0.01 0.004     0.000     0.000     0.000     0.004     0.000     0.009     0.011
               0.000     0.000     0.000
0.02 0.008     0.000     0.000     0.000     0.009     0.005     0.020     0.024
               0.005     0.005     0.005
0.03 0.011     0.000     0.000     0.000     0.016     0.019     0.033     0.036
               0.019     0.019     0.019
0.04 0.015     0.000     0.000     0.000     0.023     0.032     0.045     0.049
               0.032     0.032     0.032
0.05 0.019     0.000     0.000     0.000     0.029     0.046     0.058     0.062
               0.046     0.046     0.046
```

The output table shows the cumulative distribution of the latent (fuzzy) p-values generated at each marker position, as well as the cumultative 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 yeild a p-value less than '`pval`'. In the last row of the example output, when pval = 0.05, 4.6% of the realizations have a p-value less than 0.05 at marker-5; at marker-7 this value is 6.2%. Overall, 1.9% 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 parameters and options], page 41.

9.6 Autozyg statements

Many of the `lm_auto` and `lm_pval` statements following are also used for the location LOD scores programs.

9.6.1 Autozyg computing requests

`select [chromosome` *`I`*`] all markers [traits 1]`
: This statement selects all markers on the chromosome for the computation; if not all markers are to be used, use the next statement. If the trait for `lm_auto` is linked, '`traits 1`' is included in the statement. Omit '`traits 1`' for `lm_pval`.

`select [chromosome` *`I`*`] markers` *`J1 J2...`* `[traits 1]`
: This alternate form of the '`select markers`' statement specifies a subset of the markers. As in the previous statement, specify a linked trait for `lm_auto`, but not for `lm_pval` here.

`select unlinked trait` *`K`*
: Use this statement for `lm_auto` if the trait is unlinked. *K* is the trait number.

9.6.2 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 8. Some additional file identification statements are specific to Autozyg.

`output scores file` *`filename`*
: This statement, optional for `lm_auto`, is used to save interim scores.

`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.

`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.

`input seed file` *`filename`*
: This statement is optional for both `lm_auto` and `lm_pval`. It specifies random number seeds for L- and M- samplers.

9.6.3 Autozyg pedigree file description

Both Autozyg programs use the general MORGAN pedigree file description statements; see Section 2.5 [Pedigree file description statements], page 10.

9.6.4 Autozyg output file description

Two output file description statements are optional for `lm_auto`.

`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.

`output scores every` *`I`* `MC iterations`

If an output scores file is specified, but this statement is not present, `lm_auto` writes the scores file at the conclusion of the last iteration only (see '`set MC iterations`' statement).

9.6.5 Autozyg mapping model parameters

- For specifying the marker map, see Section 5.4.2 [genedrop mapping model parameters], page 22.
- To specify a trait map for `lm_auto`, (see Section 5.4.2 [genedrop mapping model parameters], page 22).

  The trait number specifies its position in the pedigree record; you may need to use the '`input pedigree record traits`' statement (see Section 2.5 [Pedigree file description statements], page 10) to establish the correspondence between trait numbers and integers in the pedigree record.

9.6.6 Autozyg population model parameters

- See Section 5.4.3 [genedrop population model parameters], page 23, 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` 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`'.

9.6.7 Autozyg computational parameters

- See Section 7.4 [ibddrop statements], page 36, for statements specifying the proband gametes and locus window for `lm_auto`.
- See Section 7.4 [ibddrop statements], page 36, 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 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
```

`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 data (genotypic | discrete | quantitative)`

Trait data are specified as genotypic, discrete (phenotypic), or quantitative (continuous). 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 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.

`set [component` *M*`] affected individuals` *N1 N2* `...`

`lm_pval` needs to know which members of the pedigree are affected with the disease: there are two ways to specify this. Affected individuals (for at least one component) may be named in this statement. Or, use the following statement if the affected individuals are to be determined from the pedigree data file.

`set affected individuals trait` *K*

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 2.5 [Pedigree file description statements], page 10).

9.6.8 Autozyg MCMC parameters and options

These statements which set the parameters for the MCMC algorithms, apply to both Autozyg programs, and to the location LOD score programs, `lm_lods`, `lm_markers`, `lm_bayes` and `lm_schnell` 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.

10 Estimating *ibd* Based Test Statistics by MCMC

10.1 Introduction to `lm_ibdtests`

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 (S) 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(S|Y). Four different *ibd* measures are implemented in the program. Two of these measures, Slambda and 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 (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 Basu et al. (2008) Annals of Human Genetics 72: 676-682

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 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 Basu et al (2009) Biometrics 66: 205-213;for a real data analysis using `lm_ibdtests`, see Sieh et al. 2005. Comparison of marker types and map assumptions using Markov chain Monte Carlo-based linkage analysis of COGA data. BMC Genetics 2005, 6 Suppl 1 S11)

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 the L-sampler method will be used instead of the M-sampler method with a probability of 0.5. 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 iteration number 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.

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 seed files.

For more details about the lm_ibdtest methods see Basu et al. (2008) Annals of Human Genetics 72: 676-682.

10.4 `lm_ibdtests` statements

`compute ibd statistics`
: Required.

`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.

11 Estimating Location LOD Scores by MCMC

11.1 Introduction to `lm_lods`, `lm_markers` and `lm_multiple`, `lm_bayes` and `lm_schnell`

The programs `lm_lods`, `lm_markers`, `lm_multiple`, `lm_bayes` and `lm_schnell` are referred to as "Lodscore" programs. 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 very limited in their choice of penetrance function, but this is currently under revision and will change in future releases of MORGAN.

`lm_lods` estimates location LOD scores for genotypic or discrete traits by working along the chromosome, estimating likelihood ratios between adjacent locations of the trait locus, starting from unlinked and proceeding through the linkage group to unlinked again. We have three methods of combining these local likelihood ratios into an overall LOD score method. One reduces to an eigenvalue method used by Thompson (2000: sec 9.2, P.118). Other alternatives are simply to combine the ratios from the left, or from the right. Weighted combinations do a better job (William Stewart), but we do not pursue this here as better methods are available in `lm_bayes` and `lm_markers`.

`lm_markers` and `lm_multiple` are implementations of the Lange-Sobel estimator, using our LM-sampler and the new LMM-sampler respectively. The program `lm_markers` is so-named because only the meiosis indicators at marker loci are sampled, and only conditional on the marker data. 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. Recall that the method samples meiosis indicators conditionally only on the marker data. Because of this the method can produce quite accurate LOD scores in the absence of linkage, but can be inaccurate in estimating the strength of linkage signals. As well as producing the LOD score, our current method provides a batch-means pointwise estimate of the Monte Carlo standard error of the LOD-score estimate. `lm_markers` can work with genotypic, discrete or quantitative traits.

`lm_multiple` generalizes the `lm_markers` program in various ways. In fact, from MORGAN 2.8.2 (Spring 2006), the executable `lm_markers` is compiled as a special case of the more general `lm_multiple` program. As well as including better exact computation and pedigree peeling options for use in the lod score estimator (see Section 8.3 [Exact HMM computations], page 39), the `lm_multiple` uses the new multiple-meiosis (MM) sampler in conjunction with the L-sampler. The `lm_multiple` program and MM-sampler are the work of Liping Tong (Tong & Thompson, 2008, Human Heredity 65: 142-153). Both `lm_markers` and `lm_multiple` code optionally perform exact lodscore computations on small pedigree components.

`lm_bayes` is an alternative method implemented for genotypic or discrete traits. The MCMC performance is better than for `lm_markers`, 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 (to produce an approximately uniform posterior) a second run is made to estimate the likelihood. It is important that the initial run is long enough for all points 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 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.

`lm_schnell` uses MCMC realizations of segregation indicators, conditional on marker and quantitative trait data, to estimate local likelihood ratios between alternative hypothesized trait locations. It is based on the program SCHNELL (Single CHromosome Non-Exponential Linkage Likelihoods), originally written by Greg Snow. Because `lm_schnell` uses the same local-likelihood-ratio based method of lodscore estimation as `lm_lods`, it suffers from the same disadvantages, namely extensive MCMC requirements and frequent difficulty estimating local likelihood ratios across the positions of highly polymorphic markers. However, because `lm_schnell` models a quantitative, rather than qualitative trait, MCMC mixing performance should be better. Also, uniquely among our currently released programs, `lm_schnell` models a polygenic component in addition to the major trait locus. The sampling of this component is by single-site updating, and testing of this feature has been limited. Joint updating of polygenic values is implemented in programs under development, and `lm_schnell` will be improved or replaced in future releases.

11.2 Sample parameter files for `lm_lods`, `lm_markers`, `lm_bayes` and `lm_schnell`

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, and quantitative traits, respectively. Each of these files is written for use with `lm_markers` since it is our preferred program and it can analyze genotypic, discrete, and quantitative traits. The programs `lm_lods`, `lm_bayes`, and `lm_schnell` each require slight modifications to the parameter files, and may be limited in the trait data types they can handle. These modifications are included in each of the parameter files (see commenting).

The three parameter files share the following statements:

```
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 traits 1

map trait 1 all interval proportions 0.3 0.7
```

```
    map trait 1 external recomb fracts   0.05 0.15 0.3 0.4 0.45
    set trait 1  freqs 0.5 0.5

    sample by scan
    set burn-in iterations 150
    set L-sampler probability 0.2
    set MC iterations 3000
    check progress MC iterations 1000
```

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 like in the example above.

The two '`map trait`' statements give trait locus positions at which the LOD scores should be calculated. When the trait locus is located between two markers, the trail 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 trait positions are specified to be at 30 and 70 percent of the interval. The second '`map trait`' statement allows trial trait positions located before the first marker or after the last marker to be specified; the postitions are specified explicitly in terms of recombination fractions 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 '`set trait ... 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.

The final five 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. In the '`set burn-in iterations`' statement, 100 burn-in iterations, with an unlinked trait, are requested. The L-sampler probability is set at 20 percent, which seems to be a good choice. For a detailed discussion of effects of varying L- to M-sampler ratio, see section 10.6 in Thompson (2000). The next statement requests 3000 MCMC iterations per test position of the trait locus. This is for demonstration purposes only. For real data analysis, use longer runs, on the order of 10^5 iterations per test position. The last statement tells the program to check the progress every 1000 iterations.

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 2.5 [Pedigree file description statements], page 10). The three trait data types discussed in this example are implemented by including the following statements in the parameter file discussed above.

'`ped73_ge.par`' specifies a genotypic trait with the following statements:

```
set trait data genotypic
input pedigree record trait 1 integer 3
```

'`ped73_ph.par`' specifies a phenotypic trait with the following statements:

```
set trait data discrete
input pedigree record trait 1 integer 3
set incomplete penetrance 0.05 0.6 0.95
```

Recall that for discrete data, one must specify the penetrances see Section 9.6.7 [Autozyg computational parameters], page 51.

'`ped73_qu.par`' specifies a quantitative trait with the following statements:

```
set trait data quantitative
input pedigree record trait 1 real 1

set trait 1 genotype  mean 90.0 100.0 110.0
set trait 1 residual  variance 25.0
```

When using a quantitative trait, genotypic means and residual variance must be specified. Additive variance may be needed to run `lm_schnell` and can be specified with the statement '`set trait ... additive variance`'. The default value is zero. Note that the '`set peeling by component`' statement can be used with `lm_markers` to compute LOD scores for each component separately, as well as jointly over all components; see the parameter file '`ped73_qu.par`' for this option.

The parameter files '`ped73_ge.par`' and '`ped73_ph.par`' may be used with `lm_bayes`, since the program works for genotypic and discrete traits.

With `lm_lods` and `lm_schnell`, MCMC is performed at each position, and so the number of MC iterations needed is less.

Below is a summary of the trait data types accepted for each program:

	Genotypic	**Phenotypic**	**Quantitative**	**Notes**
	ped73_ge.par	*ped73_ph.par*	*ped73_qu.par*	
lm_markers	Yes	Yes	Yes	
lm_bayes	Yes	Yes	No	
lm_schnell	No	No	Yes	Less MC Iterations Required
lm_lods	Yes	Yes	No	Less MC Iterations Required

11.3 Running lm_lods example and sample output

To run the `lm_lods` example with the phenotypic trait data, open the parameter file '`ped73_ph.par`' in the '`Lodscores/`' subdirectory. Then follow the instructions for changing the number of MC iterations. The relevant section of the parameter file is below:

```
#For actual analyses, recommended number of iterations is
#on the order of 10^5
set MC iterations 3000             #1
check progress MC iterations 1000
```

```
set global MCMC

#TO RUN LM_LODS, comment out the line marked #1 (above),
#and uncomment the line marked #2 (below). This effectively
#reduces the number of MC iterations

#Recall that lm_lods and lm_shnell require less iterations
#(order of 10^4) than other programs

#set MC iterations 300           #2
```

Recall that since MCMC is performed at each position, the number of MC iterations needed is less. Once the parameter file has been changed, run the `lm_lods` example under the subdirectory '`Lodscores`' by typing:

`./lm_lods ped73_ph.par`

The most important parts of the output are the LOD scores; these are given at the end of the output for each component (connected pedigree) at each position requested. Since there are 73 individuals in the pedigree, this example takes a while to run.

Below are the LOD scores outputs from this example (some outputs have been omitted to save space):

```
ESTIMATED LOD SCORES

 Component   1

   The largest eigenvalue        :  1.86626

   The second largest eigenvalue :  1.57587

   Cumulative from left          :  2.21620

   Cumulative from right         :  0.45122

LodScore estimates:

Trait pos #     position (Haldane cM)
  or marker         male     female         eigen       left      right

          1    -115.129   -115.129       0.04481    0.00015   -0.34546
          2     -80.472    -80.472       0.13091   -0.00410   -0.34971
          3     -45.815    -45.815       0.28046   -0.05817   -0.40378
          4     -17.834    -17.834       0.43549   -0.18552   -0.53113
          5      -5.268     -5.268       0.83469   -0.13949   -0.48510
   marker-1       0.000      0.000            NA         NA         NA
          6       3.000      3.000       1.33851    0.00175   -0.34386
          7       7.000      7.000       1.68532    0.05630   -0.28931
   marker-2      10.000     10.000            NA         NA         NA
          8      13.000     13.000       2.30193    0.17720   -0.16841
```

```
           9      17.000     17.000        2.58626    0.23244   -0.11317
    marker-3      20.000     20.000             NA         NA         NA
          10      23.000     23.000        3.46676    0.62422    0.27861
          11      27.000     27.000        3.95936    0.78769    0.44208
    marker-4      30.000     30.000             NA         NA         NA
          12      33.000     33.000        5.05419    0.97612    0.63051
          13      37.000     37.000        5.42606    1.08006    0.73445
    marker-5      40.000     40.000             NA         NA         NA
          14      43.000     43.000        6.14399    1.17323    0.82762
          15      47.000     47.000        6.39609    1.23478    0.88917
    marker-6      50.000     50.000             NA         NA         NA
          16      53.000     53.000        5.68067    1.06031    0.71470
          17      57.000     57.000        5.37402    0.98868    0.64307
    marker-7      60.000     60.000             NA         NA         NA
          18      63.000     63.000        4.32190    1.05923    0.71362
          19      67.000     67.000        3.91308    1.05841    0.71280
    marker-8      70.000     70.000             NA         NA         NA
          20      73.000     73.000        3.35744    1.01417    0.66856
          21      77.000     77.000        3.07940    1.04257    0.69696
    marker-9      80.000     80.000             NA         NA         NA
          22      83.000     83.000        2.96763    1.33124    0.98563
          23      87.000     87.000        2.79748    1.45101    1.10540
   marker-10      90.000     90.000             NA         NA         NA
          24      95.268     95.268        1.78970    1.13057    0.78496
          25     107.834    107.834        1.13899    0.95320    0.60759
          26     135.815    135.815        0.41807    0.59170    0.24609
          27     170.472    170.472        0.10067    0.43978    0.09417
          28     205.129    205.129        0.01432    0.39120    0.04559
```

Note that your output may be slightly different due to differences in the random seed. Note also that `lm_lods` does not estimate LOD scores at marker positions; the markers are included in the list of positions for reference, but the lodscore is given as `"NA"`.

As mentioned in the introduction to this chapter, there are three methods to combine the likelihood ratios (for each test position over the position to the left, and over the position to the right): the eigenvalue method, simple averaging starting from the left, and simple averaging starting from the right.

In theory, the largest real eigenvalue should be equal to 2.0. The eigenvector corresponding to the largest real eigenvalue is given as the LOD scores. However, when the second largest eigenvalue is very close to the largest one, the eigenvector can be very unstable and sometimes gives very bad LOD scores. When that happens, the “left” and “right” method, although simpler, actually perform better.

Ideally the ‘`Cumulative from left`’ and ‘`Cumulative from right`’ values at the ends of the chromosome should be log(1) or 0. In practice, the two values rarely equal one and LOD scores differ a lot for these three methods, as can be seen in the output above. This example performed a very short MCMC run. For longer runs, the LOD scores for the three methods may be more consistent with each other.

For more information regarding the MCMC parameters and diagnostic output, See Section 8.5 [MCMC parameters and options], page 41.

11.4 Running `lm_markers` examples and sample output

`lm_markers` can be run with all three parameter files in the '`Lodscores/`' subdirectory. As usual, the syntax for running the program is:

```
./lm_markers <parameter file>
```

This section describes the output obtained my using the parameter file '`ped73_qu.par`'. Note that this file contains an option for peeling by component, see commenting at the end of the parameter file for implementation. The peeling by component option allows for LOD scores to be computed for each component separately, as well as jointly over all components.

To run the example, type:

```
./lm_markers ped73_qu.par
```

The interesting part of the output is the LodScore estimates. For each test position, we have the LOD score and the standard deviation.

```
LodScore estimates by Rao-Blackwellized computation:

 Trait pos #     position (Haldane cM)
   or marker         male      female       LodScore    StdErr

           1     -115.129    -115.129         0.0312    0.0006
           2      -80.472     -80.472         0.0532    0.0012
           3      -45.815     -45.815         0.0561    0.0031
           4      -17.834     -17.834        -0.0989    0.0078
           5       -5.268      -5.268        -0.3636    0.0160
    marker-1        0.000       0.000        -0.5626    0.0248
           6        3.000       3.000        -0.5639    0.0248
           7        7.000       7.000        -0.6048    0.0280
    marker-2       10.000      10.000        -0.6660    0.0316
           8       13.000      13.000        -0.6474    0.0208
           9       17.000      17.000        -0.6788    0.0133
    marker-3       20.000      20.000        -0.7628    0.0147
          10       23.000      23.000        -0.4993    0.0183
          11       27.000      27.000        -0.2416    0.0292
    marker-4       30.000      30.000        -0.0863    0.0407
          12       33.000      33.000         0.5250    0.0352
          13       37.000      37.000         0.9361    0.0275
    marker-5       40.000      40.000         1.1317    0.0182
          14       43.000      43.000         0.9977    0.0165
          15       47.000      47.000         0.7870    0.0153
    marker-6       50.000      50.000         0.5955    0.0153
          16       53.000      53.000         0.4909    0.0152
          17       57.000      57.000         0.2737    0.0151
    marker-7       60.000      60.000         0.0359    0.0181
          18       63.000      63.000        -0.2081    0.0103
```

```
            19      67.000      67.000          -0.6324    0.0066
      marker-8      70.000      70.000          -1.0908    0.0080
            20      73.000      73.000          -0.6078    0.0125
            21      77.000      77.000          -0.3453    0.0109
      marker-9      80.000      80.000          -0.2890    0.0083
            22      83.000      83.000          -0.0640    0.0064
            23      87.000      87.000           0.1769    0.0068
     marker-10      90.000      90.000           0.3237    0.0107
            24      95.268      95.268           0.4945    0.0070
            25     107.834     107.834           0.5835    0.0037
            26     135.815     135.815           0.3911    0.0014
            27     170.472     170.472           0.1810    0.0004
            28     205.129     205.129           0.0812    0.0001
```

For more information regarding the MCMC parameters and diagnostic output, See Section 8.5 [MCMC parameters and options], page 41.

11.5 Running lm_bayes examples and sample output

If you have been following the tutorial in order, you have altered the parameter file 'ped73_ph.par' in order to reduce the number of MCMC iterations for running the lm_lods example. Since lm_bayes **DOES NOT** perform MCMC at each position, it is not advisable to reduce the number of iterations like we did in the lm_lods example. Before running the example for lm_bayes, ensure that the parameter file 'ped73_ph.par' has the larger MC iterations option selected. The relevant section of your parameter file should look like this, with the 'set MC iterations 300' commented out and the 'set MC iterations 3000' not commented out:

```
#For actual analyses, recommended number of iterations is
#on the order of 10^5
set MC iterations 3000            #1
check progress MC iterations 1000
set global MCMC

#TO RUN LM_LODS, comment out the line marked #1 (above),
#and uncomment the line marked #2 (below). This effectively
#reduces the number of MC iterations

#Recall that lm_lods and lm_shnell require less iterations
#(order of 10^4) than other programs

#set MC iterations 300
```

Under the subdirectory 'Lodscores/', run the lm_bayes example by typing:

```
./lm_bayes ped73_ph.par
```

The results from lm_bayes are the LOD scores toward the end of the output. Two methods of computing the LOD scores are available: (1) count realizations of locations sampled to estimate the posterior probability (crude) and (2) Rao-Blackwellized estimator (R-B). The latter is the preferred method.

```
LodScore estimates:

Trait pos #     position (Haldane cM)      pseudo      freq        LodScore
 or marker          male     female         prior   visited      crude        R-B

         0     unlinked   unlinked      0.020178       116         NA         NA
         1     -115.129   -115.129      0.020560       100    -0.0726    -0.0080
         2      -80.472    -80.472      0.021170       143     0.0700    -0.0203
         3      -45.815    -45.815      0.023718       165     0.0828    -0.0681
         4      -17.834    -17.834      0.035627       122    -0.2250    -0.2402
         5       -5.268     -5.268      0.060088       122    -0.4520    -0.4592
   marker-1       0.000      0.000            NA        NA         NA         NA
         6        3.000      3.000      0.089923       127    -0.6097    -0.6299
         7        7.000      7.000      0.098887       150    -0.5786    -0.6763
   marker-2      10.000     10.000            NA        NA         NA         NA
         8       13.000     13.000      0.106325       102    -0.7776    -0.7095
         9       17.000     17.000      0.105926       111    -0.7393    -0.7091
   marker-3      20.000     20.000            NA        NA         NA         NA
        10       23.000     23.000      0.067969        93    -0.6234    -0.5206
        11       27.000     27.000      0.043461       105    -0.3765    -0.3216
   marker-4      30.000     30.000            NA        NA         NA         NA
        12       33.000     33.000      0.022509       118    -0.0401    -0.0447
        13       37.000     37.000      0.014511        87     0.0182     0.1340
   marker-5      40.000     40.000            NA        NA         NA         NA
        14       43.000     43.000      0.009146       157     0.4751     0.3324
        15       47.000     47.000      0.007793        82     0.2625     0.4142
   marker-6      50.000     50.000            NA        NA         NA         NA
        16       53.000     53.000      0.010758       101     0.2130     0.3376
        17       57.000     57.000      0.017022       121     0.0922     0.1807
   marker-7      60.000     60.000            NA        NA         NA         NA
        18       63.000     63.000      0.025113       117    -0.0913    -0.0129
        19       67.000     67.000      0.027450        45    -0.5449    -0.1115
   marker-8      70.000     70.000            NA        NA         NA         NA
        20       73.000     73.000      0.027063        83    -0.2729    -0.1503
        21       77.000     77.000      0.026552        67    -0.3576    -0.1337
   marker-9      80.000     80.000            NA        NA         NA         NA
        22       83.000     83.000      0.023659        96    -0.1513    -0.0742
        23       87.000     87.000      0.019262        55    -0.3039     0.0179
  marker-10      90.000     90.000            NA        NA         NA         NA
        24       95.268     95.268      0.012658        81     0.0465     0.2023
        25      107.834    107.834      0.011441        87     0.1215     0.2475
        26      135.815    135.815      0.014517        76    -0.0406     0.1441
        27      170.472    170.472      0.017645        90    -0.0520     0.0588
        28      205.129    205.129      0.019069        81    -0.1314     0.0248
```

11.6 Running `lm_schnell` example and sample output

Recall that `lm_schnell` works with quantitative traits only, and that because MCMC is performed at each position, the number of MC iterations needed is less. Since the number of MC iterations required is less, we must edit the parameter file '`ped73_qu.par`' to reflect this (just like we did when running the `lm_lods` example). Instructions are located near the end of the parameter file, and are given below:

```
#For actual analyses, recommended number of iterations is
#on the order of 10^5
set MC iterations 3000            #1
check progress MC iterations 1000
set global MCMC

#TO RUN LM_SCHNELL, comment out the line marked #1 (above),
#and uncomment the line marked #2 (below). This effectively
#reduces the number of MC iterations

#Recall that lm_lods and lm_shnell require less iterations
#(order of 10^4) than other programs

#set MC iterations 300            #2
```

Under the subdirectory '`Lodscores/`', run the example with the following command:

```
./lm_schnell ped73_qu.par
```

Since additive variance is not specified in the parameter file, the default value 0 is used. This example takes a while to finish.

```
 ESTIMATED LOD SCORES

  Component 1

    The largest eigenvalue:          2.38058

    The second largest eigenvalue:   1.84655

    Cumulative from left   :         3.47316

    Cumulative from right :          0.28792

    LOD scores:
    position     female        male       eigen        left       right
           0 -115.12925 -115.12925     0.04135     0.01177    -0.52896
           1  -80.47190  -80.47190    -0.04939     0.04327    -0.49746
           2  -45.81454  -45.81454    -0.20917     0.10753    -0.43320
           3  -17.83375  -17.83375    -0.72080     0.20552    -0.33520
           4   -5.26803   -5.26803    -1.42463    -0.11293    -0.65366
           5    3.00000    3.00000    -1.90715    -0.16779    -0.70852
           6    7.00000    7.00000    -2.20055    -0.23907    -0.77979
           7   13.00000   13.00000   -2.78293   -0.06788   -0.60860
           8   17.00000   17.00000   -3.33181   -0.14982   -0.69055
           9   23.00000   23.00000   -4.36919    0.01748   -0.52324
          10   27.00000   27.00000   -4.61200    0.09763   -0.44309
          11   33.00000   33.00000   -4.76603    0.93494    0.39421
          12   37.00000   37.00000   -4.92469    1.19967    0.65894
          13   43.00000   43.00000   -5.47771    1.22628    0.68555
          14   47.00000   47.00000   -5.74826    1.36085    0.82013
          15   53.00000   53.00000   -6.35674    0.74257    0.20184
          16   57.00000   57.00000   -6.15321    0.48212   -0.05860
          17   63.00000   63.00000   -6.19531   -0.03028   -0.57100
          18   67.00000   67.00000   -6.27084   -0.56559   -1.10631
          19   73.00000   73.00000   -4.94832   -0.61375   -1.15447
          20   77.00000   77.00000   -4.35057   -0.29956   -0.84028
          21   83.00000   83.00000   -3.34770    0.11962   -0.42110
          22   87.00000   87.00000   -2.86960    0.22810   -0.31263
          23   95.26803   95.26803   -1.58121    0.97457    0.43385
          24  107.83375  107.83375   -0.68635    1.08899    0.54826
          25  135.81454  135.81454   -0.36267    0.83854    0.29781
          26  170.47190  170.47190   -0.24994    0.67770    0.13698
          27  205.12925  205.12925   -0.08595    0.61535    0.07462
```

The ‘`eigen`’, ‘`left`’ and ‘`right`’ columns have the same interpretation as in `lm_lods`.

For more information regarding the MCMC parameters and diagnostic output, See Section 8.5 [MCMC parameters and options], page 41.

11.7 Location LOD scores statements

New statements for these programs include maps for test positions, and parameters for some additional MCMC algorithms.

11.7.1 Location LOD scores computing requests

- For the ‘`select`’ statement for your MCMC simulation, See Section 9.6.1 [Autozyg computing requests], page 50. Select all or some of the markers and ‘`traits 1`’ (this is the trait to be assigned varying test positions).

11.7.2 Location LOD scores file identification statements

All Lodscore programs use the general MORGAN file identification statements (see Section 2.3 [File identification statements], page 8) and the Autozyg rescue file statements (see Section 9.6.2 [Autozyg file identification statements], page 50). 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.

11.7.3 Location LOD scores pedigree file description

All Lodscore programs use the general MORGAN pedigree file description statements; See Section 2.5 [Pedigree file description statements], page 10. One additional statement is optional for `lm_markers` and `lm_schnell`.

`input pedigree record traits` *K1 K2...* `reals` *I1 I2...*

> This statement is analogous to '`input pedigree record traits` *K1 K2...* `integers` *I1 I2...*' (see Section 2.5 [Pedigree file description statements], page 10), when the trait is quantitative, rather than discrete.

11.7.4 Location LOD scores output file description

All Lodscore programs use the Autozyg output file description statements; See Section 9.6.4 [Autozyg output file description], page 50.

11.7.5 Location LOD scores mapping model parameters

- See Section 5.4.2 [genedrop mapping model parameters], page 22, for statements specifying the genetic map for the markers.

`map [chromosome` *I*`] [gender (M | F)] trait` *K* `all interval proportions` *X1 X2...*

> Interval proportions specify the proportional genetic distance between markers for the trial positions for the trait. The same ratios are used between each marker pair, regardless of the inter-genetic distance (in cM).

`map [chromosome` *I*`] [gender (M | F)] trait` *K* `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)] trait` *K* `(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.

11.7.6 Location LOD scores population model parameters

- See Section 5.4.3 [genedrop population model parameters], page 23, for statements specifying the allele frequencies for the markers and traits, and See Section 9.6.6 [Autozyg population model parameters], page 51, for statements specifying marker names.

11.7.7 Location LOD scores computational parameters

- See Section 7.4 [ibddrop statements], page 36, for setting the sampler seeds.
- See Section 9.6.7 [Autozyg computational parameters], page 51, for specifying the marker data.
- See Section 9.6.7 [Autozyg computational parameters], page 51, for specifying the trait data as genotypic, quantitative or discrete and for specifying penetrances when trait data are discrete.

`set pseudo-priors` *`X1 X2`* `...`
: This statment is optional for `lm_bayes`. The number of pseudo-priors is the number of trial trait 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.

11.7.8 Location LOD scores MCMC parameters and options

- All statements described in Section 9.6.8 [Autozyg MCMC parameters and options], page 53, for specifying the MCMC parameters are also used for the location LOD scores programs.

As with the Autozg 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_markers`, the total MCMC run length is the sum of the number of burn-in iterations and main iterations. For `lm_lods` and `lm_schnell`, the total MCMC run length is the number of burn-in iterations, plus the product of the number of test positions for the trait, (see Section 11.7.5 [Location LOD scores mapping model parameters], page 69), and the number of 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 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.

12 Polygenic Modeling of Quantitative Traits by EM Algorithm

12.1 Introduction to PolyEM programs

PolyEM is a set of programs to evaluate the likelihood and compute MLEs for polygenic models of quantitative traits by EM algorithms. 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 polygenic peeling, which does not extend to looped pedigrees, and by a matrix method which does not extend to large pedigrees.
- `unibig`: An extension of `univar` to big pedigrees that implements a method compute the likelihood for large looped pedigrees too.
- `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 (e.g., any statements with covariances do not apply to `univar` or `unibig`).

12.2 Sample `multivar` parameter file

The pedigree file for PolyEM programs 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 trait 1
select 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. A '`select trait`' statement is required for each trait to be modeled (up to 5 traits). This statement indicates which column of real numbers the program is to examine. In the example, the statement '`select trait 1`' indicates that the first column of real numbers is to be examined. The statement '`select trait 2 effects 1 2`' indicates that the second column of real numbers is to be examined as well. The additional '`effects 1 2`' portion of the statement is optional and indicates that two *fixed effects* (also called *covariates*) are to be modeled for trait 2. 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. 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, thePolyEm 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 12.4.2 [multivar segregation model parameters], page 75, for discussion of the order of 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 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 12.4.3 [multivar computational parameters], page 75).

12.3 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`' statment 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.

```
    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
```

12.4 `multivar` statements

12.4.1 `multivar` computing requests

`select trait` *I* `[effects` *M1*`...]`

One '`select trait`' statement is required for each trait 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.

M's are the fixed effects to be modeled for a specified trait *I*. They are the columns in which they appear in the pedigree file, with fixed effect *1* in the 4th column (usually *gender*), effect *2* in column 5, and so on.

12.4.2 `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.

12.4.3 `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.

12.4.4 `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.

12.4.5 `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 Ith 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.

13 Estimating Genetic Map from Marker Data

13.1 Introduction to `lm_map`

The program `lm_map` finds the maximum likelihood estimate (MLE) of the marker map, estimates the 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.

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 Stewart, WCL and Thompson, EA (2006) Improving estimates of genetic maps: A maximum likelihood approach. Biometrics 62, 728-734.

13.2 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" denote 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`'.

'`map_G.par`' and '`map_P.par`' have the following statements in common:

```
input pedigree file './map.ped'
input marker data file './map_[G|P].markers'

select all markers
set marker 1 2 3 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 9.6.1 [Autozyg computing requests], page 50). The '`set marker 1 2 3 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 31). The '`set marker names`' statement overrides the default behavior, which labels markers consequtively: marker-1, marker-2, etc.

The two '`map gender [] marker recomb fract`' statements specify the marker map in terms of recombination fractions.

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 9.6.8 [Autozyg MCMC parameters and options], page 53. The statement '`set MC iterations 50`' indicates how many MC iterations are to be performed at each step. 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`' 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.

Now we'll take a look at the remaining statements in '`map_P.par`':

```
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`'.

13.3 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
```

13.4 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.

13.5 `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 ny 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 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 gentypes 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 stat 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.

Concept Index

Statement Index