

MORGAN V3.0 Tutorial

Version 3.0.3
Last Updated: 20 October 2011

Table of Contents

1	Get Started	1
1.1	Overview of MORGAN	1
1.2	Get the Tutorial	2
1.3	Get and set up the examples	2
1.4	Overview of the pedigrees used in the examples	3
1.5	Structure of the MORGAN package	4
2	Common Features and File Formats	7
2.1	Command syntax	7
2.2	Parameter file	8
2.3	File identification statements	8
2.4	Limit overriding	10
2.5	Output control	10
2.6	Pedigree file	11
2.7	Pedigree file description statements	12
3	Checking Pedigree Validity	15
3.1	Introduction to <code>pedcheck</code>	15
3.2	Sample <code>pedcheck</code> parameter file	15
3.3	Running <code>pedcheck</code> examples	16
3.4	<code>pedcheck</code> statements	17
4	Computing Kinship and One- or Two-Locus Inbreeding Coefficients	18
4.1	Introduction to <code>kin</code>	18
4.2	Sample <code>kin</code> parameter file	18
4.3	Running <code>kin</code> example and sample output	19
4.4	<code>kin</code> statements	20
5	Simulating Marker and Trait Data in Pedigrees	22
5.1	Introduction to <code>genedrop</code>	22
5.2	Sample <code>genedrop</code> parameter file	23
5.3	Running <code>genedrop</code> examples and sample output	25
5.4	<code>genedrop</code> statements	26
5.4.1	<code>genedrop</code> computing requests	26
5.4.2	<code>genedrop</code> mapping model parameters	26
5.4.3	<code>genedrop</code> population model parameters	27
5.4.4	<code>genedrop</code> computational parameters	28
5.4.5	<code>genedrop</code> output pedigree options	28
5.4.6	<code>genedrop</code> output seed file options	29

6	Simulating Marker Data Conditional on Trait Data in Pedigrees	30
6.1	Introduction to <code>markerdrop</code>	30
6.2	Sample <code>markerdrop</code> parameter file – conditional on trait	30
6.3	Sample <code>markerdrop</code> parameter file – conditional on inheritance pattern	33
6.4	Running <code>markerdrop</code> examples and sample output	34
6.5	<code>markerdrop</code> statements	35
6.5.1	<code>markerdrop</code> computing requests	36
6.5.2	<code>markerdrop</code> mapping model parameters	36
6.5.3	<code>markerdrop</code> population model parameters	37
6.5.4	<code>markerdrop</code> computational parameters	37
6.5.5	<code>markerdrop</code> input file options	38
7	Estimating <i>a priori</i> <i>ibd</i> Probabilities by Monte Carlo	39
7.1	Introduction to <code>ibddrop</code>	39
7.2	Sample <code>ibddrop</code> parameter file	39
7.3	Running <code>ibddrop</code> example and sample output	40
7.4	<code>ibddrop</code> statements	42
8	Using MCMC to Estimate Parameters of Interest in Pedigree Data	44
8.1	Specifying inheritance	44
8.2	Genetic model	44
8.3	Exact HMM computations	45
8.4	Single and multiple meiosis LM-samplers	46
8.5	MCMC computational options	47
8.6	MCMC parameter statements	49
9	Estimating Conditional <i>ibd</i> Probabilities by MCMC	51
9.1	Introduction to <code>lm_auto</code> , <code>gl_auto</code> and <code>lm_pval</code>	51
9.2	Sample <code>lm_auto</code> parameter file	52
9.3	Running <code>lm_auto</code> example and sample output	55
9.4	Sample <code>gl_auto</code> parameter file	57
9.5	Running <code>gl_auto</code> example and sample output	59
9.6	Sample <code>lm_pval</code> parameter file	60
9.7	Running <code>lm_pval</code> example and sample output	60
9.8	Autozyg statements	61
9.8.1	Autozyg computing requests	61
9.8.2	Autozyg file identification statements	62
9.8.3	Autozyg pedigree file description	63
9.8.4	Autozyg output file description	63
9.8.5	Autozyg mapping model parameters	63
9.8.6	Autozyg population model parameters	64

9.8.7	Autozyg computational parameters	64
9.8.8	Autozyg MCMC parameters and options	65
10	Estimating <i>ibd</i> Based Test Statistics by MCMC.....	67
10.1	Introduction to <code>lm_ibdtests</code> and <code>civil</code>	67
10.2	Sample <code>lm_ibdtests</code> parameter file	68
10.3	Sample <code>lm_ibdtests</code> output	69
10.4	Sample <code>civil</code> parameter file	70
10.5	Sample <code>civil</code> output	73
10.6	<code>lm_ibdtests</code> and <code>civil</code> statements	74
11	Estimating Location lod Scores by MCMC	76
11.1	Introduction to <code>lm_linkage</code> , <code>lm_bayes</code> , <code>lm_twoqt1</code> , and <code>gl_lods</code>	76
11.2	Sample parameter files for <code>lm_linkage</code> and <code>lm_bayes</code>	77
11.3	Running <code>lm_linkage</code> examples and sample output	80
11.4	Running <code>lm_bayes</code> examples and sample output	82
11.5	Running <code>lm_twoqt1</code> examples and sample output.....	83
11.6	Parameter files for the <code>gl_lods</code> program	86
11.7	Running <code>gl_lods</code> examples and sample output	90
11.8	Location lod scores statements	91
11.8.1	Location lod scores computing requests	91
11.8.2	Location lod scores file identification statements	91
11.8.3	Location lod scores pedigree file description.....	92
11.8.4	Location lod scores output file description	92
11.8.5	Location lod scores mapping model parameters	92
11.8.6	Location lod scores population model parameters	93
11.8.7	Location lod scores computational parameters	93
11.8.8	Location lod scores MCMC parameters and options.....	94
12	Polygenic Modeling of Quantitative Traits by EM Algorithm.....	96
12.1	Introduction to PolyEM programs.....	96
12.2	Sample <code>multivar</code> parameter file.....	96
12.3	Running <code>multivar</code> example and sample output	98
12.4	<code>multivar</code> statements.....	100
12.4.1	<code>multivar</code> computing requests	100
12.4.2	<code>multivar</code> segregation model parameters	100
12.4.3	<code>multivar</code> computational parameters	101
12.4.4	<code>multivar</code> computational options	102
12.4.5	<code>multivar</code> output options.....	102

13 Estimating Genetic Maps from Marker Data	104
.....	
13.1 Introduction to <code>lm_map</code>	104
13.2 Sample <code>lm_map</code> parameter file	104
13.3 Running <code>lm_map</code> with genotypic data	107
13.4 Running <code>lm_map</code> with phenotypic data	107
13.5 <code>lm_map</code> statements	108
References	110
Concept Index	112
Statement Index	116

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 3.0.3 which was released in November 2011. It is available for download through the **MORGAN** 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 15. `kin` computes kinship and inbreeding coefficients for members of the pedigree, see [Chapter 4 \[Computing Kinship and One- or Two-Locus Inbreeding Coefficients\]](#), page 18.
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 22. `markerdrop` simulates marker data at loci linked to a trait locus, see [Chapter 6 \[Simulating Marker Data Conditional on Trait Data in Pedigrees\]](#), page 30. `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 39.
3. Programs using Markov chain Monte Carlo (MCMC) techniques: MORGAN’s MCMC programs are split into two sections, ‘Autozyg’ and ‘Lodscore’. These programs typically have the prefix ‘`lm_`’ indicating that they use the MCMC LM-sampler. In fact, most of these programs now have the option to use the improved multiple-meiosis LM-sampler. A brief introduction to the MCMC sampling techniques employed by MORGAN can be found in [Chapter 8 \[Using MCMC to Estimate Parameters of Interest in Pedigree Data\]](#), page 44.

The ‘Autozyg’ and ‘Lodscore’ programs may be categorized in four subsets:

- The Autozyg programs `lm_auto`, `gl_auto` and `lm_pval`, estimate conditional gene *ibd* probabilities; see [Chapter 9 \[Estimating Conditional IBD Probabilities by MCMC\]](#), page 51.
- The Autozyg programs `lm_ibdtests` and `civil` realize inheritance conditional on genetic marker data and uses these realizations to estimate *ibd*-based test statistics for linkage detection; see [Chapter 10 \[Estimating IBD Based Test Statistics by MCMC\]](#), page 67.
- The Lodscore programs `lm_linkage`, `lm_bayes` and `lm_twoqtl`, estimate multi-point lod scores; see [Chapter 11 \[Estimating Location lod Scores by MCMC\]](#), page 76.
- The Autozyg program `lm_map` realizes inheritance conditional on genetic marker data, and uses these realizations in the estimation of genetic maps; see [Chapter 13 \[Estimating Genetic Maps from Marker Data\]](#), page 104.

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 96.

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

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.

See [\[Concept Index\]](#), page 112 for: MORGAN, overview of MORGAN.

1.2 Get the Tutorial

This tutorial is available on-line at

<http://www.stat.washington.edu/thompson/Genepi/MORGAN/Morgan.shtml#tut>

Several formats of this tutorial are also available to download for off-line reading or printing. (These may not be available with the initial online release of the tutorial. The html, PDF, and plain text versions are recommended.)

- [Single HTML file](#).
- [Gzipped multiple HTML files](#) (one file per section).
- [Hyperlinked PDF file](#).
- [Gzipped Postscript file](#).
- [Gzipped info file](#).
- [Plain text file](#).

See [\[Concept Index\]](#), page 112 for: how to get the tutorial.

1.3 Get and set up the examples

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

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

1. Download the examples (gzipped tar files) for MORGAN 3.0.3. [morgan-examples_V30.tar.gz](#)
2. Unpack the examples by typing the following command in a shell window,

```
tar zxvf morgan-examples_V30.tar.gz
```

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

```
gunzip -c morgan-examples_V30.tar.gz | tar xvf -
```

This will produce a 'MORGAN_V30_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.)

- Use ‘Makefile’ to establish links under the ‘MORGAN_V30_Examples’ directory to the MORGAN programs. A link under the ‘MORGAN_V30_Examples’ directory serves as a shortcut to a MORGAN program installed elsewhere.

Before making links, you first need to edit the ‘Makefile’ (using your favorite text editor, for instance `vim` or `nano`) in the ‘MORGAN_V30_Examples’ directory to make sure the paths to your MORGAN programs and those to the ‘MORGAN_V30_Examples’ directory are correct. Most often, it is necessary to change the ‘MORGANDIR’ and ‘EXAMPLEDIR’ statements to reflect the locations of the MORGAN files on your system and the examples, respectively. Here is the relevant part of the ‘Makefile’,

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

MORGANDIR = ~/morgan/MORGAN_V302_Release
EXAMPLEDIR = `pwd`
BINDIR = ~/bin

# Note: the paths may happen to be same for MORGANDIR and
# EXAMPLEDIR. In general they are different:
# MORGANDIR is where MORGAN is installed on your system
# EXAMPLEDIR is the MORGAN_V30_Examples directory you have made
# (we have used the BASH command `pwd` to automate this)
# BINDIR is your bin directory
# BINDIR is needed only if you prefer to link to executables from
# your bin directory, rather than running from executables in
# a current directory.
```

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

```
make help
```

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

```
make links
```

Notes for Microsoft Windows users:

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

See [\[Concept Index\]](#), page 112 for: how to get the examples.

1.4 Overview of the pedigrees used in the examples

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

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

The other pedigree in ‘ped73.ped’, located under ‘MORGAN_V30_Examples’, consists of three components and 73 individuals: component one has 47 individuals over 6 generations, component two 11 individuals over 3 generations, and component three 15 individuals over 3 generations. In general, individuals from later generations are observed. The three components are displayed in ‘ped47.pdf’, ‘ped11.pdf’ and ‘ped15.pdf’, which are located in the subdirectory ‘PedInfo’.

See [Concept Index], page 112 for: Tutorial for lod score examples with ‘ped73.ped’.

1.5 Structure of the MORGAN package

It is not necessary to read this section in order to use MORGAN, to run the examples, or to modify them for your own use. However, for those who wish to modify MORGAN code, or to understand MORGAN more fully, it will be useful to have information on the directory structure, the README documentation, and the GOLD-standard documentation, Makefiles, and examples. These are therefore described in this section, updated for the released version of MORGAN 3.0.3.

1. README documentation files

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

1. README files in the main MORGAN directory

These include ‘README_readme’, ‘README_MORGAN’, ‘README_install’, and ‘README_relnotes’.

- ‘README_readme’ describes the various README files throughout MORGAN.
- ‘README_MORGAN’ lists the MORGAN programs and describes briefly the analysis done by each program. It also lists the MORGAN 3.0.3 directories and libraries.
- ‘README_relnotes’ contains a summary of the changes and additions in recent releases of MORGAN.
- ‘README_install’ contains instructions for installing MORGAN executables.

In some MORGAN releases there may be additional main-directory README files.

2. README files in main program directories

The main program directories of MORGAN 3.0.3 are PedComp, Genedrop, Autozyg, Lodscore, and PolyEM. Each main program directory contains its own ‘README_userdoc’. This describes the inputs to be prepared for the programs, and the various program options. Most of this information is now included in the tutorial, but the README files may contain more detail in some cases.

Each of the main program directories contains a file ‘README_convert2_3.0’ which specifies the changes one must make in order to convert parameter files from MORGAN 2.9 to 3.0.

The Library Subroutine directories do not contain README files.

3. README files in Gold and Test subdirectories.

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

2. The subroutine library directories

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

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

The libraries can be divided broadly into four groups:

1. Lowest-level libraries required by all programs
 - Stuff: Routines for printing, allocating, freeing
 - Pars: Routines for processing MORGAN parameter statements
2. Low-levels libraries performing various groups of functions
 - CMF: A set of routines mainly for matrix manipulation, originally translated from the FORTRAN CM library
 - Peel: Routines for pedigree peeling computations
 - Rans: Routines for random number generation
3. Main libraries supporting genetic analysis programs
 - Pedchk: Routines for checking validity of input pedigrees. Routines for the 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.
4. Extra specialist libraries
 - 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.

3. The main program directories

The main program directories of MORGAN 3.0.3 are PedComp, Genedrop, Autozyg, Lodscore, and PolyEM. When MORGAN is installed these directories contain the following executables:

- PedComp: `pedcheck`, `kin`, `translink`
- Genedrop: `genedrop`, `ibddrop`, `markerdrop`
- Autozyg: `lm_auto`, `gl_auto`, `lm_pval`, `lm_ibdtests`, `civil`, `lm_map`
- Lodscore: `lm_linkage`, `lm_bayes`, `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.

4. GOLD-STANDARD directories, Makefiles and examples

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

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

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

See [\[Concept Index\]](#), page 112 for: structure of the MORGAN package, README documentation files, MORGAN program libraries, MORGAN subroutine libraries, MORGAN Gold standards.

2 Common Features and File Formats

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

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

See [\[Concept Index\]](#), page 112 for: white space.

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:

```
./progrname parfile
```

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

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

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

```
progrname parfile
```

but the form *./progrname* 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:

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

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

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

`oxtr` Output extra 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.

See [\[Concept Index\]](#), page 112 for: MORGAN files, command syntax, command line options, file type codes.

2.2 Parameter file

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

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

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_V30_Examples/Pedcheck'`.

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

A brief description of the most commonly used parameter file statements follows in the next section. For a complete and more detailed description of MORGAN statements, please see the sections of this tutorial relevant to specific MORGAN programs and the documentation that comes with MORGAN in the files `'README_userdoc'` in the various program subdirectories. See [\[Concept Index\]](#), page 112 for: parameter file, parameter statements.

2.3 File identification statements

Within the parameter file, file names are delimited with single or double quotation marks (`'` or `"`). File names submitted on the command line are not delimited with quotation marks.

In a parameter file, either of the two statements below would identify ‘pedchk.ped’ as the pedigree file to be read.

```
input pedigree file "pedchk.ped"  
input pedigree file 'pedchk.ped'
```

The most commonly used file identification statements are:

input pedigree file *filename*

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

input extra file *filename*

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

input marker data file *filename*

Marker data, such as marker allele frequencies, map distances between markers and individuals’ genotypes, can be included in the parameter file itself or in a separate file, called the marker data file. This statement is used when the marker data are not included within the parameter file. The marker file contains the ‘set marker data’ statements. Marker data are used by Autozyg programs. See [Section 9.8.7 \[Autozyg computational parameters\], page 64](#).

input seed file *filename*

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

output [overwrite] pedigree file *filename*

The output pedigree file is required by **genedrop**. Other programs also check for errors in the pedigree. If there are errors that the program is able to correct or if there are requested changes to the pedigree file format, the new pedigree data is written to this file.

output [overwrite] seed file *filename*

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

output [overwrite] score file *filename*

The output score (or scores) file is used by several programs to output numerical results, typically in a format for input to another analysis program.

output [overwrite] extra file *filename*

The output extra file is used by some programs to output additional results.

Note that with MORGAN 3.0 several overwrite options have been added for output files, including pedigree and output scores files. Previously output scores were appended to

existing output, if the file already existed, leading to confusion. This remains so, unless the the overwrite option is used. Users should be cautious is using (and in not using) the overwrite option, and should, if using the option, be careful to copy previous output to another filename should they wish to retain it.

See [Concept Index], page 112 for: file names, pedigree file, extra file, overwrite file options, marker data file, seed file.

2.4 Limit overriding

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

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

allow component size *N*

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

allow observed individuals *N*

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

allow pedigree size *N*

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

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

See [Concept Index], page 112 for: limit overriding, limit statements, allow statements,

2.5 Output control

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

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

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

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

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

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

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

set printlevel *N*

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

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

set debug main

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

set debug *libname*

The **set debug *libname*** statement is available for each library, where *libname* is one of **cmf**, **markers**, **nghds**, **pars**, **pedchk**, **peel**, **quant**, **rans**, **sample**, **stuff** or **twoqt1**, corresponding to the relevant library name. The statement will cause additional information to be printed to **stdout** as coded in the sub-routine files of that specific library.

set debug level

The **set debug level** statement can be used in any program to, in principle, set the required level of additional debugging statements. However, currently only the **lm_twoqt1** program and 'TwoQTL' library include code making use of the debug level.

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

See [\[Concept Index\]](#), page 112 for: output control, redirect output, display options, GLUT runtime display, debug control, printlevel control, output warnings (W).

2.6 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. Gender is coded as an integer, such that '1', '2' and '0' represent male, female, and unsexed, respectively.

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

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

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

See [Concept Index], page 112 for: pedigree file, parameter statements in pedigree files, file separator, whitespace, names of individuals, observed individuals, unobserved individuals.

2.7 Pedigree file description statements

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

allow pedigree size *N*

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

input pedigree size *N*

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

input pedigree record names 3 [integers *I*] [reals *J*]

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

input pedigree record (father mother | mother father)

This statement specifies the order of parental names. 'father mother' is the default.

input pedigree record gender (present | absent)

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

`input pedigree record observed (absent | present)`

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

`input pedigree record traits K1 K2... integers I1 I2...`

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

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

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

Below are the first several lines of the sample pedigree file, 'ped73.ped' in 'MORGAN_V30_Examples'.

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

```
*****
```

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

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

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

The 6 integers following gender and the real number in the final column represent individual data. Lack of an 'input pedigree record traits integers' statement would imply that

the first integer following gender corresponds to trait 1, the second to trait 2, etc. However, these parameter statements are typically provided in the parameter file, not in the pedigree file.

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

See [\[Concept Index\]](#), [page 112](#) for: pedigree file descriptions, pedigree size, pedigree record format, gender, pedigree trait data order.

3 Checking Pedigree Validity

3.1 Introduction to pedcheck

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

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

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

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

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

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

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

See [\[Concept Index\]](#), page 112 for: `pedcheck` introduction, pedigree validity, component.

3.2 Sample pedcheck parameter file

Files for `pedcheck` may be found in the ‘`Pedcheck`’ subdirectory of ‘`MORGAN_V30_Examples`’. Below is the sample parameter file ‘`check.par`’ for `pedcheck`:

```
set printlevel 5
input pedigree file 'check.ped'
```

```

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

```

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

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

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

See [\[Concept Index\]](#), page 112 for: `pedcheck` sample parameter file, component, pedigree component, overwrite file options.

3.3 Running `pedcheck` examples

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

```
./pedcheck check.par
```

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

`./pedcheck imp.par`

runs on input pedigree file ‘`imp.ped`’. The pedigree contains an individual who is his own ancestor.

`./pedcheck empty.par ped sex.ped`

runs with an empty parameter file, apart from a ‘`set printlevel 5`’ request. The input pedigree file ‘`sex.ped`’ is 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?

See [Concept Index], page 112 for: `pedcheck` examples.

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 output pedigree file, see Section 2.3 [File identification statements], page 8.
- For describing the format of the pedigree file, see Section 2.7 [Pedigree file description statements], page 12.

`(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.

See [Concept Index], page 112 for: `pedcheck` statements, `pedcheck` statements, pedigree options, gender, pedigree order, component, pedigree component.

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

See [Concept Index], page 112 for: `kin` introduction, kinship coefficient, inbreeding coefficient, *ibd*,

4.2 Sample kin parameter file

Files for `kin` may be found in the 'IBD' subdirectory of 'MORGAN_V30_Examples'. Below is a sample `kin` parameter file, 'jv_rep_kin.par'.

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

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

component, you may first run `pedcheck` to determine which individuals are assigned which component numbers.

See [Concept Index], page 112 for: kin sample parameter file.

4.3 Running kin example and sample output

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

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

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 freq	g4link 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 freq	g4link prob
441	.000	.06250
	.010	.05885
	.040	.04905
	.050	.04614
	.100	.03388
	.180	.02060
	.300	.01008
	.500	.00391

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

See [\[Concept Index\]](#), page 112 for: running `kin` example, `kin` sample output.

4.4 kin statements

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

`compute [component M] kinship coefficient N1 N2...`

This statement names one or more pairs of pedigree members for which the kinship coefficient is to be computed.

`compute [component M] inbreeding coefficient N1...`

This statement names one or more pedigree members for whom the inbreeding coefficient is to be computed.

`compute [component M] two-locus inbreeding coefficient N1...`

This statement requests the computation of two-locus inbreeding coefficients, i.e. the probability of *ibd* at both loci, for the named individual. For the recombination frequencies at which the coefficients are computed, see the following statement.

`set recombination frequencies X1 X2...`

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

See [\[Concept Index\]](#), page 112 for: `kin` statements, component, pedigree component, kinship coefficient, inbreeding coefficient, recombination.

5 Simulating Marker and Trait Data in Pedigrees

See [\[Concept Index\]](#), page 112 for: simulating marker data, simulating trait data.

5.1 Introduction to *genedrop*.

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

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

Because founders of a pedigree are assumed to be unrelated, a unique identifier, a *founder genome label* or *founder gene 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
<i>non-genetic model</i>	YES	YES
<i>polygenic model</i>	YES	NO
<i>major gene model</i>	NO	YES
<i>mixed model</i>	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 2.6 \[Pedigree file\]](#), page 11, for details.

See [\[Concept Index\]](#), page 112 for: *genedrop* introduction, quantitative trait, polygenic model, major gene model, mixed model, non-genetic model, founder genome labels, founder gene labels, seeds for data simulation, additive variance, unobserved individuals.

5.2 Sample genedrop parameter file

Files for `genedrop` may be found in the ‘Simulation’ subdirectory of ‘MORGAN_V30_Examples’. The example here refers to ‘ped73_gdrop.par’.

The seed file is used to store the random seeds used in the simulations. Occasionally one will want to use the same seed with multiple runs, but most often one will want to use new seeds so as to obtain different output with each run. The seed file contains one or more statements like ‘`set marker seeds 0xde5e8d39`’. For more about the way `genedrop` handles seeds See [Section 5.4.4 \[genedrop computational parameters\]](#), page 28.

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

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

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

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

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

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

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

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

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

The above statement indicates a marker map on chromosome 1, with 10 equally spaced markers, each at a distance of 10 (Haldane) centiMorgans from the preceding one. Note

that the number of markers is inferred from this statement. The trait locus is between markers 5 and 6 on chromosome 1, at a distance of 5 cM to marker 5.

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

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

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

Marker allele frequencies are set by the following lines:

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

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

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

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

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

```
set trait 1 allele freqs 0.5 0.5

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

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

```
output pedigree record founder 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 output for all (observed and unobserved) individuals. Founder gene labels indicate, for all non-founders, which founder alleles were passed to the individual. For the trait variables, the latent founder gene labels, the trait locus genotype, and the additive and residual contributions to the trait value are given. Latent trait variables will precede the trait value in the output file.

See [Concept Index], page 112 for: `genedrop` sample parameter file, seed file, additive variance, residual variance, founder gene labels.

5.3 Running `genedrop` examples and sample output

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

The command to run the first example is:

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

When running the `genedrop` example, notice ‘`marker.seed`’ is specified as both the input and output seed file. If a ‘`overwrite`’ option is not included in the ‘`output seed file`’ statement, successive runs will generate warnings (W), but this is not a concern. Recall from the previous section that, by default, `MORGAN` appends the new output seeds to the existing seed file at the end of each run. In the next run, the last (most recent) seed will be used. To avoid this warning (and an ever-growing seed file), either use the ‘`overwrite`’ when outputting the seeds (see the previous section [Section 5.2 \[Sample `genedrop` parameter file\], page 23](#)), or manually edit the seed file removing earlier lines.

Since the function of `genedrop` is to simulate marker and trait data, it, unlike other `MORGAN` programs, always creates and outputs a pedigree file. The output file ‘`gdrop.oped`’ is structured similarly to the input file ‘`ped73.ped`’, with one individual per record (line). However, the output file contains additional columns and does not include the parameter statements found at the top of the input file. The first four items are the individual’s name, the names of the parents, and gender. If no additional 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 (FGL) for markers precede the marker genotypes and the trait FGL precede the trait values (or the trait latent variables, if these are requested).

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

See [Concept Index], page 112 for: running `genedrop` examples, `genedrop` sample output, seeds for data simulation.

5.4 `genedrop` statements

See [Concept Index], page 112 for: `genedrop` statements.

5.4.1 `genedrop` computing requests

`simulate [chromosome I] markers`

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

`simulate traits K1`

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

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

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

`map tlocs L1... unlinked`

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

See [Concept Index], page 112 for: `genedrop` computing requests.

5.4.2 `genedrop` mapping model parameters

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

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

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

This statement is required if simulated trait loci are to be linked to markers; i.e., it is not required if no trait loci or only unlinked trait loci are to be simulated.

The statement specifies the location of each trait locus with respect to one of the marker loci. Thus, the number of trait loci listed in the statement must be equal to the number of markers listed and to the number of distances (or recombination fractions) listed. The trait locus will follow the corresponding marker locus (to the right, so to speak) at the distance specified. To simulate a trait locus that precedes all marker loci, list marker '0' in the statement. For example, with 'map tlocs 3 2 marker 6 0 distances 5 4', trait loci 3 and 2 will be placed 5 cM to the right of marker 6 and 4 cM to the left of marker 1, respectively.

See [\[Concept Index\]](#), page 112 for: `genedrop` mapping model parameters, gender-specific maps, Haldane map function, Kosambi map function.

5.4.3 `genedrop` population model parameters

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

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

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

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

`set normalized allele frequencies`

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

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

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

`set traits K1 ... additive variance X`

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

`set traits K1 ... residual variance X`

This statement is like the preceding one. The environmental contribution to the trait is set using this statement.

See [\[Concept Index\]](#), page 112 for: `genedrop` population model parameters, allele frequencies, additive variance, residual variance.

5.4.4 genedrop computational parameters

set marker seeds *H1 H2*

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

set trait seeds

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

See [Concept Index], page 112 for: **genedrop** computational parameters, seeds for data simulation, simulating marker data, simulating trait data.

5.4.5 genedrop output pedigree options

output pedigree record founder gene labels

When this option is selected, each record contains a pair of founder 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 gene labels are determined by the simulated meiosis indicators.

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

output pedigree record trait latent variables

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

output pedigree record unobserved variables

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

When this option is not selected, unobserved individuals take on default values; the genotype at each locus represented as '0 0', the founder 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.

See [Concept Index], page 112 for: **genedrop** output pedigree options, founder gene labels, meiosis indicators, inheritance indicators, unobserved individuals.

5.4.6 genedrop output seed file options

output (marker | trait) seeds only

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

See [Concept Index], page 112 for: **genedrop** output seed file, seeds for data simulation.

6 Simulating Marker Data Conditional on Trait Data in Pedigrees

6.1 Introduction to `markerdrop`

`markerdrop` simulates marker data at markers linked to a hypothetical trait locus. The user must specify whether marker data simulation is to be conditional on a trait model and trait data (the `trait` option) or as a (possibly partial) specification in the pedigree file of the inheritance at the trait location (the `inheritance` option). The choice of a trait model or an inheritance pattern will dictate which additional parameter statements must (or may) be included in the parameter file. For the `trait` option, the pedigree file contains trait data; for the `inheritance` option, the pedigree file contains meiosis (`inheritance`) indicators. See [Section 8.1 \[Specifying inheritance\], page 44](#).

- If marker data simulation is to be conditional on a trait model, parameters must be provided for trait locus allele frequencies using `'set tlocs allele frequencies'`, for genotypic penetrances using `'set trait for tlocs incomplete penetrances'`, and for the map position of the trait locus using a `'map'` statement see [Section 6.5 \[markerdrop statements\], page 35](#). There must be only one mapping statement for the trait locus; from this statement the trait locus number (name) is deduced. Phenotypic trait data are provided as affection status of each individual in the pedigree file. An inheritance pattern at the trait locus is simulated from the trait data; this becomes the trait model on which markers are simulated.
- If marker data simulation is to be conditional on an inheritance pattern at the trait locus, the partially specified segregation pattern at the trait locus is provided in the pedigree file using meiosis indicators. For more information on meiosis indicators, see [Section 6.5.1 \[markerdrop computing requests\], page 36](#) and [Chapter 8 \[Using MCMC to Estimate Parameters of Interest in Pedigree Data\], page 44](#). Location of meiosis indicators in the pedigree file can be specified using the `'input pedigree record'` statement. Again, specification of a map position for the trait locus using a `'map'` statement is required.

See [\[Concept Index\], page 112](#) for: `markerdrop` introduction, trait model, incomplete penetrances, inheritance indicators, meiosis indicators.

6.2 Sample `markerdrop` parameter file – conditional on trait

Files for `markerdrop` may be found in the `'Simulation'` subdirectory of `'MORGAN_V30_Examples'`. The sample parameter file `'ped73_mdrops_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.

Note that the `'ped73_mdrops_trait.par'` parameter file contains a `'set printlevel'` statement. MORGAN programs will produce varying levels of output given the print level. We recommend setting the print level to 5 for initial testing purposes.

Many of the statements for simulation of the markers conditional on trait data are similar to those used in `genedrop`: See [Section 5.2 \[Sample genedrop parameter file\], page 23](#). However, rather than simulating trait loci or trait data, these are provided to the `markerdrop` program.

The relevant section of the file is:

```

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

set trait 2 data discrete

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

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

101 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
102 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
202 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
301 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
304 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
.
.
.

```

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

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

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

The `markerdrop` program uses the `'set markers 10 data'` statement to specify which individuals and at which loci marker data are required. This is the same statement used by

other programs in the analysis of marker data; see, for example [Section 9.8.7 \[Autozyg computational parameters\]](#), page 64. Marker data are specified for each marker locus as a pair of integer alleles, and ‘0’ indicates a missing value. For `markerdrop` any non-zero value will indicate that the data are to be observed. Typically, one may enter regular marker data, in order to generate other marker data with the same missingness pattern. Alternatively, as here, a ‘1’ may be used to indicate that the corresponding marker data are observed. Note that individual ‘302’ is the first observed member in this data set, and is observed for all 10 markers.

The parameter file ‘`ped73_mdrops_trait.par`’ uses the pedigree file ‘`ped73.ped`’, which is found in the ‘`MORGAN_V30_Examples`’ directory. The file format section and first few lines of the pedigree data section of this file are below.

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

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

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

```
select trait 2
```

Now we see also in the parameter file

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

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

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

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

See [\[Concept Index\]](#), page 112 for: `markerdrop` parameter file – conditional on trait, set `printlevel` statement.

6.3 Sample markerdrop parameter file – conditional on inheritance pattern

The sample parameter file, ‘ped73_mdrops_inhe.par’, requests simulation of marker data conditional on an inheritance pattern. The relevant section of the file is:

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

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

101 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
102 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
202 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
301 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
304 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
.
.
.
```

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

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

The parameter file ‘ped73_mdrops_inhe.par’ uses pedigree file ‘ped73.ped’. The file format section and first few lines of the pedigree data section of this file are below.

```
input pedigree record names 3 integers 7 reals 1
*****
101 0 0 1 0 0 0 -1 -1 0 999.5
```

```

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

```

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

The connection to these inheritance data is through the statement

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

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

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

For more information on `markerdrop` options see [Section 6.5 \[markerdrop statements\]](#), [page 35](#).

See [\[Concept Index\]](#), [page 112](#) for: `markerdrop` parameter file – conditional on inheritance pattern.

6.4 Running markerdrop examples and sample output

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

```

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

```

if your PATH includes your current directory.

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

```
./markerdrop ped73_mdrip_trait.par > mdrip_trait.out
```

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

```
./markerdrop ped73_mdrip_inhe.par > mdrip_inhe.out
```

After running `markerdrop` with the parameter file 'mdrip_inhe.par', and the pedigree file 'ped73.ped' (as in the above example), the output file 'mdrip_inhe.out' is generated. Some sections of this output file are given below. Note that similar output would be generated using 'ped73_mdrip_trait.par'.

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

```

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

```

.....

assigned FGL in all listed individuals:

trait locus, followed by 10 marker loci

```

101  2 1  2 1  2 1  2 1  2 1  2 1  2 1  2 1  2 1  2 1  2 1  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  0 0
102  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
201  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
202  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
2010 0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
301  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0  0 0
302  4 2  4 3  5 6  3 3  6 1  1 2  3 3  1 4  1 6  1 4

```

.....

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

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

See [\[Concept Index\]](#), page 112 for: running `markdrop` examples, `markdrop` output, founder gene labels.

6.5 markerdrop statements

See [\[Concept Index\]](#), page 112 for: `markdrop` statements.

6.5.1 markerdrop computing requests

`markerdrop` always requires the following statement:

`simulate markers`

This statement requests that markers are to be simulated. Whether the simulation is conditional on a trait model or on an inheritance pattern is inferred from the following statements.

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

`select trait K`

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

`select inheritance H`

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

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

This statement establishes the correspondence of traits to trait loci; it is used when the trait option is selected.

`set inheritance H1... tlocs L1...`

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

See [\[Concept Index\]](#), page 112 for: `markerdrop` computing requests marker simulation, marker simulation using trait, marker simulation using meiosis indicators.

6.5.2 markerdrop mapping model parameters

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

This statement is required for `markerdrop` if more than one marker is to be simulated. It specifies the marker map (optionally a sex-specific map), in units of genetic distance (cM), marker position (cM), or recombination fraction. If

distance is selected, `markerdrop` will expect one fewer values than the number of markers, as these are intermarker distances. If position is expected, the same number of values as markers will be expected, as these are the positions of the markers relative to some zero point to the left of marker 1. If Kosambi is not specified, the Haldane mapping function is used to convert between genetic distance and recombination fraction.

```
map [gender (F | M)] tlocs K marker J ( [Kosambi] distance | recombination
fraction ) X
```

This statement is required for `markerdrop`; it tells the program which trait locus to use in the simulation of marker data and gives a location for the trait locus, either as a map distance or recombination fraction, following the marker listed in the statement. As with `genedrop`, to simulate a trait locus position that precedes all markers, list the marker number as '0'.

See [\[Concept Index\]](#), page 112 for: `markerdrop` mapping model parameters

6.5.3 markerdrop population model parameters

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

This statement specifies trait locus allele frequencies. Traits 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 [chromosome I] marker names M1 M2...
```

This statement specifies marker names in the order of their position along the chromosome. Default names are marker-1, marker-2, etc.

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

Marker allele frequencies are specified using this statement. A marker can have up to 100 alleles and all allele frequencies must be listed. For each marker, the allele frequencies should sum to between 0.9999 and 1.0001. Otherwise they are automatically normalized and a warning message will be issued. Multiple markers can be included in a single statement if they have the same number of alleles with the same frequencies.

See [\[Concept Index\]](#), page 112 for: `markerdrop` population model parameters, trait allele frequencies, marker names, marker allele frequencies.

6.5.4 markerdrop computational parameters

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

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

```
set trait K data discrete
```

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

As with `genedrop`, marker seeds and trait seeds can be specified or the default values can be used, See [Section 5.4.4 \[genedrop computational parameters\]](#), page 28.

See [\[Concept Index\]](#), page 112 for: `markerdrop` computational parameters, penetrance, incomplete penetrance, discrete trait, trait data, marker seeds, trait seeds.

6.5.5 markerdrop input file options

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

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

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

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

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

See [\[Concept Index\]](#), page 112 for: `markerdrop` input file options, pedigree record format.

7 Estimating *a priori ibd* Probabilities by Monte Carlo

See [\[Concept Index\]](#), page 112 for: *a priori ibd* probabilities, identity by descent, *ibd*.

7.1 Introduction to *ibddrop*

ibddrop estimates probabilities of gene identity by descent, *ibd*, (such as kinship, inbreeding, or multi-gene identities) by Monte Carlo in the absence of data. Given the pedigree and a genetic map, *ibddrop* simulates meioses indicators and scores them to estimate the *ibd* probabilities among a set of gametes.

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.

See [\[Concept Index\]](#), page 112 for: *ibddrop* introduction, *ibd* pattern, meiosis indicators, inheritance indicators.

7.2 Sample *ibddrop* parameter file

Files for *ibddrop* may be found in the 'IBD' subdirectory of 'MORGAN_V30_Examples'. The sample parameter file for *ibddrop* is 'jv_rep_ibd.par'.

```
set printlevel 5
input pedigree file 'jv_rep.ped'

simulate markers
simulate tloc 1

map          markers  distances 44.6 44.6 11.2 11.2
map tlocs 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 reason for this specification is that the same specification may then be used in `lm_auto`, where simulation is conditional on marker and (optionally) trait data. See [Chapter 9 \[Estimating Conditional IBD Probabilities by MCMC\]](#), page 51.

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 42).

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 a multilocus *ibd* probability, the statement ‘`set locus window`’ can be used to specify number of loci to score jointly. `ibddrop` has limited functionality for computing multilocus probabilities, it can only examine two gametes to determine whether or not the two are *ibd*. For 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 52: `lm_auto` has the option of scoring more general patterns of gene identity over multilocus windows. See [\[Concept Index\]](#), page 112 for: `ibddrop` sample parameter file, Haldane map function, proband gametes, seeds for sampler, seed file.

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 tlocs 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  tloc-1 marker-3 marker-4 marker-5  label
1 1 1 1  .0290  .0293  .0285  .0284  .0295  .0298  0
1 1 1 2  .0271  .0298  .0285  .0294  .0288  .0283  1
1 1 2 1  .0144  .0126  .0130  .0146  .0135  .0140  3
1 1 2 2  .0095  .0107  .0106  .0093  .0092  .0089  4
1 1 2 3  .0249  .0258  .0278  .0273  .0280  .0268  5
1 2 1 1  .0693  .0644  .0664  .0654  .0659  .0633  6
1 2 1 2  .0063  .0053  .0056  .0060  .0055  .0052  7
1 2 1 3  .0599  .0605  .0585  .0585  .0597  .0585  8
1 2 2 1  .0693  .0693  .0698  .0696  .0708  .0712  9
1 2 2 2  .0495  .0479  .0489  .0490  .0490  .0471  10
1 2 2 3  .1406  .1384  .1338  .1372  .1363  .1392  11
1 2 3 1  .1376  .1368  .1401  .1364  .1374  .1391  12
1 2 3 2  .0251  .0263  .0297  .0255  .0265  .0279  13
1 2 3 3  .0956  .0958  .0961  .0976  .0954  .0958  14
1 2 3 4  .2418  .2472  .2427  .2459  .2447  .2451  15
```

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

The *ibd* patterns are scored for each locus separately; there is a column for each of the five markers and one for the trait locus.

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.

See [Concept Index], page 112 for: running `ibddrop` example, `ibddrop` sample output, *ibd* pattern.

7.4 `ibddrop` statements

- Use the ‘`simulate markers`’ statements to specify simulation of markers and one linked or unlinked trait, for each of one or more chromosomes (see Section 5.4.1 [genedrop computing requests], page 26). For convenience these key statements are repeated below.

- Use ‘map’ statements to specify the marker and trait locus maps (see [Section 5.4.2 \[genedrop mapping model parameters\]](#), page 26). Note also one additional ‘map’ statement below.

Note that `ibddrop` does not simulate or use marker or trait data. The statements are used only to specify the map of the loci at 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 55), where simulation is conditional on marker and trait data.

The additional `ibddrop` statements are:

`simulate markers`

This statement specifies that markers are to be simulated. The number of markers is inferred from the marker map.

`simulate tloc L`

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

`map tlocs L1 ... unlinked`

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

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

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

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

`set [chromosome I] locus window K`

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

`set sampler seeds H1 H2`

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

`set MC iterations I`

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

See [\[Concept Index\]](#), page 112 for: `ibddrop` statements, proband gametes, meiosis indicators, inheritance indicators, seeds for sampler.

8 Using MCMC to Estimate Parameters of Interest in Pedigree Data

See [Concept Index], page 112 for: MCMC introduction, Markov chain Monte Carlo.

8.1 Specifying inheritance

See [References], page 110, for details of the cited papers.

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

The set of all meiosis indicators is denoted $\mathbf{S} = (S_{ij})$ where

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

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

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

See [Concept Index], page 112 for: specifying inheritance, meiosis indicators, inheritance indicators, founder gene labels, founder genome labels, inheritance vector, *ibd*.

8.2 Genetic model

See [References], page 110, for details of the cited papers.

In MORGAN there are three basic genetic data types. These are: be *genotypic*, typically used for marker data; *discrete*, a data type for binary data requiring specification of *incomplete*

penetrances; and *quantitative*, using a Gaussian penetrance with specification of genotypic means and residual variance.

As yet, loci are either multiallelic marker loci assumed observed without error, or trait loci which may have general penetrance functions but 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 in [Tho77]. For zero-loop pedigrees (see [Tho76]), these peeling routines are used by the `lm_map` program which allows for errors in marker data. For general pedigrees, they are not yet released, as they are still in process of testing.

2. Penetrance functions and trait models:

Liability classes been implemented for the discrete-trait penetrance model in `lm_linkage` and `lm_bayes`. Penetrances for each liability class are read from an input file using the ‘`input extra data file S`’ parameter statement.

Additionally, an age-based penetrance function for a qualitative 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 program `lm_twoqt1` allows two (linked or unlinked) quantitative trait loci to contribute additively or epistatically to a single trait [STW07]. A polygenic component may also be included. Two-locus penetrances may be specified as additive, with a genotypic mean for each trait genotype for each locus. Alternatively, a matrix array of 2-locus genotypic means may be specified, allowing for epistasis [SW07].

With more these more complex trait models, including those of `lm_twoqt1` [STW07], a more general specification of traits is required. From MORGAN V3.0, completely new structures have been introduced, separating traits (phenotypes) from trait loci (“tlocs”). Traits may be affected by genotypes at several tlocs; the genotypes at a tloc may affect several traits.

See [Concept Index], page 112 for: genetic model, penetrance.

8.3 Exact HMM computations

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

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

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

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

See [Concept Index], page 112 for: exact computations, HMM computations.

8.4 Single and multiple meiosis LM-samplers

See [References], page 110, for details of the cited papers.

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

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

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

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

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

The new LM-sampler can also be used in the programs `lm_auto` and `gl_auto`, and in the program `lm_twoqt1`. All these MORGAN 3.0 programs sample inheritance patterns conditional on marker data for use in subsequent lod score or *ibd* computations. They all have the option to use either the old (single-meiosis) or new (multiple-meiosis) LM-sampler.

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

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

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

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

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

See [Concept Index], page 112 for: LM-sampler, pedigree peeling, multiple meiosis sampler, block Gibbs sampler, L-sampler, M-sampler, L-sampler probability.

8.5 MCMC computational options

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

At each MCMC iteration, MORGAN selects a locus (with L-sampler) or set of meioses (with M-sampler) to update. Two different selection methods are available: sample by step and sample by scan. If 'sample by scan' is chosen, all loci or meioses are updated one-at-a-time in a predetermined random order. This option is the default. If 'sample by step' is

chosen, a single locus or meiosis is randomly selected for updating at each iteration. The sampling method selected applies to the entire MCMC run, including burn-in, pseudo-prior computation and main iterations.

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

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

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

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

There are three components to the MCMC diagnostic output:

1. Average total log-probability of segregations:

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

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

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

3. Recombination counts for map intervals

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

In these diagnostic scores, for the programs `lm_pval` and `lm_linkage` only marker loci and marker map intervals are included in these diagnostic scores. For `lm_auto`, the trait locus (designated ‘0’) is included in the correct position, if it is included in the MCMC, while

the `gl_auto` program requires a null (no-data) unlinked trait locus. This null locus may or may not be included in the MCMC sampling: see ‘set MCMC markers only’ in [Section 9.8.8 \[Autozyg MCMC parameters and options\]](#), page 65.

See [\[Concept Index\]](#), page 112 for: MCMC options, sequential imputation, locus-by-locus setup, sample by step, sample by scan, burn-in, MC iterations, pseudo-prior iterations.

8.6 MCMC parameter statements

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

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

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

`use I sequential imputation realizations for setup`

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

`set MC iterations I`

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

`set burn-in iterations I`

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

`sample by (scan | step)`

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

`set L-sampler probability X`

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

`compute scores every I iterations`

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

check progress *I* MC iterations

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

set global MCMC

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

set limit for exact computation *I1*

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

See [\[Concept Index\]](#), page 112 for: MCMC parameter statements, sequential imputation, locus-by-locus sampling, meiosis indicators, inheritance indicators, MC iterations, burn-in, L-sampler, M-sampler, sample by scan, sample by step.

9 Estimating Conditional *ibd* Probabilities by MCMC

See [Concept Index], page 112 for: conditional *ibd* probabilities, identity by descent, *ibd*.

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

See [References], page 110, for details of the cited papers.

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

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

`gl_auto` also uses either the old or new LM-sampler to realize *ibd* configurations from their conditional distribution given the marker data. By default, a null (no-data) unlinked ‘trait’ locus is also included: optionally this may be omitted Section 9.8.8 [Autozyg MCMC parameters and options], page 65. Rather than estimating *ibd* probabilities, `gl_auto` outputs realizations of *ibd* configurations directly to an output file (using the MORGAN output scores file). The output may either be of founder genome labels (FGL or ‘gl’) or of the meiosis indicators that determine the FGL. The output is in a compact format where only changes of FGL or meiosis indicators are recorded, together with the positions of these changes. Positions are in terms of marker indices in the original marker data file, even where markers are subselected for MCMC. These output *ibd* graphs may be used for subsequent analyses of trait data on the pedigrees, without further reference to the pedigree structure or marker data. For further details see [Tho11].

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

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

Additional programs using latent p-values are under development, including programs for the distribution of latent lod scores obtained in MCMC sampling (`lm_fuzlod`), p-values and randomized tests based on latent lod score statistics (`lm_fzplod`), and randomized confidence sets for the location of a trait locus (`lm_fzconf`). The methods are described by Thompson in [Tho08]. The program `civil` (see [Section 10.1 \[Introduction to `lm_ibdtests` and `civil`\], page 67](#)) also uses latent p-values [DT09].

See [\[Concept Index\], page 112](#) for: Autozyg programs, `lm_auto` introduction, `gl_auto` introduction, `lm_pval` introduction, Markov chain Monte Carlo, autozygosity, inheritance indicators, meiosis indicators, identity by descent, descent graph and *ibd* graph, *ibd*, latent p-values, LM-sampler, multiple meiosis sampler.

9.2 Sample `lm_auto` parameter file

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

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

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

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

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

set tloc 1 allele freqs .95 .05

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

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

set window patterns 0 4
```

```

set locus window 3

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

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

```

The ‘select’ statement is analogous to `genedrop`’s ‘simulate’ statement (see [Section 5.4.1 \[genedrop computing requests\]](#), page 26). The statements first specify that all markers will be used. Then the trait (‘1’) for which data will be analyzed is identified and connected to a `tloc` (‘1’).

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

The ‘map’ statements specify the marker map and trait position in terms of genetic distances (centiMorgan). In this example there are five markers with gender-specific maps. The trait locus position is measured from the marker to its left. In this example, the trait locus for males is between markers 2 and 3 at a distance of 12.8 cM to the left of marker 2 (See [Section 5.4.2 \[genedrop mapping model parameters\]](#), page 26. The ‘set markers’ statements specify the number and frequency of alleles for each marker. In the example, the first four markers each have six alleles (labeled 1–6) with frequencies 0.2, 0.2, 0.4, 0.1, 0.06 and 0.04. The fifth marker has four alleles with frequencies 0.3, 0.2, 0.3 and 0.1. The trait locus has two alleles; alleles ‘1’ and ‘2’ have frequencies 0.95 and 0.05, respectively.

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

The ‘set window patterns’ and ‘set locus window’ statements instruct `lm_auto` to compute the probabilities that the gametes named in the ‘set proband gametes’ statement have a particular *ibd* pattern (also called *state*) jointly across several loci. The ‘set locus window’ statement specifies the number of loci to be examined simultaneously, in this case 3. This statement was discussed briefly in the `ibddrop` example: See [Section 7.3 \[Running ibddrop example and sample output\]](#), page 40. The probabilities in the ‘IBD’ column of the

output specifies whether one of the specified set of patterns holds ('1') or not ('0') at each of the three loci across the window.

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

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

Probabilities of IBD patterns

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

pattern	marker-1	marker-2	trait-1	marker-3	marker-4	marker-5	label
1 1 1 1	.0287	.0298	.0310	.0273	.0287	.0298	0
1 1 1 2	.0290	.0275	.0292	.0282	.0302	.0305	1
1 1 2 1	.0132	.0135	.0138	.0140	.0139	.0132	3

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

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

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

In the example, '531 1' refers to the paternal (1) haplotype of individual '531'. The first statement requests scoring both haplotypes of 531, the maternal (0) haplotype of 331, and the paternal (1) haplotype of 333. Note that currently the number of proband gametes to be scored jointly is limited to 10. See [Section 7.4 \[ibddrop statements\], page 42](#), for more discussion of the 'set proband gametes' statement.

As with all of MORGAN's MCMC-based programs, the user can specify the desired number of MC iterations using the 'set MC iterations' statement, the desired number of burn-in iterations using 'set burn-in iterations', and the probability that the L-sampler is selected instead of the M-sampler using 'set L-sampler probability'. In this example, 2000 sampling iterations are to be performed, using the L-sampler 20 percent of the time. These iterations are preceded by burn-in iterations. Because the number of burn-in iterations

is not specified, `lm_auto` will use the default value of 10 percent of the number of main iterations. In practice, one would run the MCMC sampler much longer than 2000 iterations (on the order of 10^5). The ‘multiple meiosis’ sampler is requested and the ‘set global MCMC’ statement is not used, so MCMC will be performed separately for each pedigree component. For further details of these statements: See [Section 8.6 \[MCMC parameter statements\]](#), page 49.

See [\[Concept Index\]](#), page 112 for: `lm_auto` sample parameter file, trait data, genotypic trait, gender-specific maps, marker data file, window patterns, proband gametes, L-sampler, M-sampler, burn-in.

9.3 Running `lm_auto` example and sample output

The syntax for running a MORGAN program is:

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

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

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

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

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

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

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

Probabilities of IBD for pattern set for windows of 3 loci

```

Proband gamete set 1

Pattern set:  0  4

   IBD  wndw 1  wndw 2  wndw 3  wndw 4
0 0 0   .4955  .4635  .4630  .5410
0 0 1   .0810  .0815  .0520  .0740
0 1 0   .0345  .0415  .0375  .0430
0 1 1   .1410  .0545  .0550  .0280
1 0 0   .0495  .0515  .1520  .1125
1 0 1   .0150  .0110  .0190  .0180
1 1 0   .0280  .1295  .0930  .1085
1 1 1   .1555  .1670  .1285  .0750

```

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

Probabilities of IBD patterns

```

Proband gamete set 1:  541 1  541 0

pattern marker-1 marker-2 trt-geno marker-3 marker-4 marker-5  label
1 1      .7000      .8760      .9580      .8165      .6570      .4670      0
1 2      .3000      .1240      .0420      .1835      .3430      .5330      1

```

Interpretation of these results is similar to that of `ibddrop`. See [Section 7.3 \[Running `ibddrop` example and sample output\], page 40](#). 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 40](#). Recall that in `ibddrop`, the values in the ‘IBD’ column of the output indicate whether the two gametes specified in the ‘`set proband gametes`’ statement are *ibd* (indicated by a ‘1’) or not (indicated by a ‘0’). With `lm_auto`, the user can specify additional *ibd* patterns of interest over two or more gametes. In this example, the parameter file ‘`jv_rep_auto.par`’ includes the statement ‘`set window patterns 0 4`’, which indicates

that we are interested in *ibd* patterns ‘0’ and ‘4’, corresponding to ‘1 1 1 1’ and ‘1 1 2 2’, respectively, as discussed in the previous section. That is, we would like to know the probability that either all four gametes are *ibd* or that the first two are *ibd* and the second two are *ibd*, but the pairs are not *ibd* with one another for each window of three loci. Consequently, interpretation of the ‘IBD’ column of the `lm_auto` output is as follows. The row headed by ‘0 0 0’ gives probabilities that the gametes do not follow either of the two *ibd* patterns of interest at all three loci for each window. The row headed by ‘0 0 1’ gives probabilities that the gametes do not follow either of the two *ibd* patterns of interest at the first two loci in the window, but at the third loci either all four gametes are *ibd* or the first two are *ibd* and the last two are *ibd*, but the pairs are not *ibd* with one another.

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

window	loci
wndw 1	marker 1, marker 2, trait
wndw 2	marker 2, trait, marker 3
wndw 3	trait, marker 3, marker 4
wndw 4	marker 3, marker 4, marker 5

For more information regarding the MCMC parameters and diagnostic output, See [Section 8.5 \[MCMC computational options\], page 47](#).

See [\[Concept Index\], page 112](#) for: running `lm_auto` example, `lm_auto` sample output, *ibd* pattern, proband gametes.

9.4 Sample `gl_auto` parameter file

The parameter file of the `gl_auto` example is in the ‘IBD’ subdirectory of ‘MORGAN_V30_Examples’. Like the earlier `markerdrop` example (see [Section 6.2 \[Sample markerdrop parameter file – conditional on trait\], page 30](#)), the `gl_auto` uses the 3-component pedigree with a total of 73 individuals, ‘ped73.ped’, and the corresponding 10-marker data, ‘ped73.marker.missing’. Both these files are in the main ‘MORGAN_V30_Examples’ directory; that is ‘.’ relative to the parameter file.

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

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

set printlevel 5                # Include everything in the output file.

select trait 1
input pedigree record trait 1 integer 7    # dummy (0) trait values
set trait 1 tloc 11                # Connect trait and tloc
```

```

map tloc 11 unlinked                                # Trait locus is unlinked.
set tloc 11 allele freqs 0.5 0.5

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

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

## For real analyses, recommended number of iterations is of order 105
set MC iterations 2000                               # For golds and checks only.
set burn-in iterations 15                            # For golds and checks only.

# Specify what type of output is desired.
# Specify the desired scoring interval.

output founder genome labels
output scores every 30 scored MC iterations
output scores file './ped73_glauto.scor'

```

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

The Monte Carlo requests are those used by most of the MCMC-based programs: See [Section 8.6 \[MCMC parameter statements\], page 49](#). Unlike earlier examples, the multiple meiosis sampler is used, and sampling is (by default) by pedigree component as ‘global MCMC’ is not requested. A limit of 12 meioses is set for exact computation; in fact even the smallest 11-member pedigree component has 14 meioses, so MCMC will be done on each of the three components. Other MCMC requests are standard and similar to those used in the `lm_auto` example. Note again that many more MCMC scans (and associated burn-in) would be done in a real example.

The final three parameter statements are specific to the `gl_auto` program. This program outputs *ibd* graphs to an output scores file, so the file name, the output requested (meiosis indicators or founder genome labels) and the scoring frequency are given. Here every MCMC iteration is ‘scored’ (the default), but *ibd* graphs are computed and output only every 30 iterations.

See [\[Concept Index\], page 112](#) for: `gl_auto` sample parameter file, multiple meiosis sampler, founder genome labels, exact computation, *ibd* graph.

9.5 Running `gl_auto` example and sample output

The syntax for running a MORGAN program is:

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

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

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

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

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

The first few lines of the file are shown as

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

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

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

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

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

Some important points are that, first, there are programs to process and use this *ibd* graph output; the user should not be concerned with the details. However, it is important that the user knows how many graphs they have generated, and whether they are done by component or globally. It is recommended to separate the graphs by component. Next it should be seen

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

See [Concept Index], page 112 for: running `gl_auto` example, `gl_auto` sample output.

9.6 Sample `lm_pval` parameter file

Files for `lm_pval` may be found in the ‘TraitTests’ subdirectory of ‘MORGAN_V30_Examples’. The parameter file, ‘`ped73_pval.par`’ is similar to the parameter file used for `lm_auto`. An abbreviated version of ‘`ped73_pval.par`’ is given below:

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

input pedigree record trait 1 integer 3
select trait 1

input seed file '../sampler.seed'

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

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

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

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

See [Concept Index], page 112 for: `lm_pval` sample parameter file.

9.7 Running `lm_pval` example and sample output

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

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

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

```
Combined distribution of fuzzy p-values, by locus:
pval maxim marker-1 marker-2 marker-3 marker-4 marker-5 marker-6 marker-7
           marker-8 marker-9 marker-10
```

0.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		0.000	0.000	0.000				
0.01	0.002	0.015	0.008	0.000	0.001	0.002	0.000	0.002
		0.000	0.000	0.000				
0.02	0.006	0.025	0.019	0.000	0.002	0.002	0.000	0.007
		0.000	0.000	0.000				
0.03	0.008	0.035	0.030	0.003	0.006	0.003	0.000	0.017
		0.000	0.000	0.000				
0.04	0.012	0.045	0.041	0.007	0.009	0.004	0.000	0.028
		0.000	0.000	0.000				
0.05	0.015	0.055	0.051	0.011	0.013	0.006	0.002	0.039
		0.000	0.000	0.000				

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

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

For more information regarding the MCMC parameters and diagnostic output: See [Section 8.5 \[MCMC computational options\]](#), page 47.

See [\[Concept Index\]](#), page 112 for: running `lm_pval` example, `lm_pval` sample output, fuzzy p-value.

9.8 Autozyg statements

Many of the `lm_auto` and other statements following are also used for thelocation lod scores programs. See [Section 11.8 \[Location lod scores statements\]](#), page 91.

See [\[Concept Index\]](#), page 112 for: Autozyg statements, `lm_auto` statements, `gl_auto` statements, `lm_pval` statements.

9.8.1 Autozyg computing requests

```
select [chromosome I] all markers
```

This statement selects all markers on the chromosome for the computation; if not all markers are to be used, use the next statement.

```
select [chromosome I] markers J1 J2 ...
```

This alternate form of the ‘`select markers`’ statement specifies a subset of the markers.

```
select trait K
```

This statement names the selected trait, and is used in both `lm_auto` and `lm_pval`. For the former, this names the trait to be used in subsequent mappings

to trait loci. For the latter, it is used to determine disease affection status of individuals.

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

This statement establishes the correspondence between traits and trait loci in `lm_auto`. Note that this statement is not applicable for `lm_pval`.

`map tlocs L1 ... unlinked`

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

`check marker consistency [only]`

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

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

	No error is present	Some error(s) present
No statement	No termination	Terminates at first error
<code>check markers consist</code>	No termination	After checking all markers
<code>check mark consist only</code>	After checking all markers	After checking all markers

See [\[Concept Index\]](#), [page 112](#) for: Autozyg computing requests consistency (Mendelian) of marker data.

9.8.2 Autozyg file identification statements

All of the general MORGAN file identification statements can be used with the '`Autozyg`' programs. For a list of these statements, see [Section 2.3 \[File identification statements\]](#), [page 8](#). Some additional file identification can be used by '`Autozyg`' programs:

`output score file 'filename'`

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

`input rescue file 'filename'`

A rescue file may be used to continue an `lm_auto` run instead of restarting at the beginning. This file contains intermediate data, which is periodically saved when an output rescue file has been specified in a preceding run.

`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.

See [Concept Index], page 112 for: Autozyg file identification statements, score file, rescue file.

9.8.3 Autozyg pedigree file description

Both Autozyg programs use the general MORGAN pedigree file description statements; see Section 2.7 [Pedigree file description statements], page 12.

See [Concept Index], page 112 for: Autozyg pedigree file description.

9.8.4 Autozyg output file description

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

`output (founder genome labels | meiosis indicators)`

This statement is expected by '`gl_auto`'. It tells the program whether its output is to be in the form of founder genome labels or meiosis indicators.

`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 scored MC iterations`

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

See [Concept Index], page 112 for: Autozyg output file description, founder genome labels.

9.8.5 Autozyg mapping model parameters

- For specifying the marker map, see Section 5.4.2 [genedrop mapping model parameters], page 26.
- To specify a trait map for `lm_auto`, see Section 5.4.2 [genedrop mapping model parameters], page 26.

The trait number specifies its position in the pedigree record; you may need to use the '`input pedigree record traits`' statement (see Section 2.7 [Pedigree file description statements], page 12) to establish the correspondence between trait numbers and integers in the pedigree record.

See [Concept Index], page 112 for: Autozyg mapping model parameters.

9.8.6 Autozyg population model parameters

- See [Section 5.4.3 \[genedrop population model parameters\]](#), page 27, for statements specifying the allele frequencies for the markers and traits.

set [chromosome *I*] marker names *M1 M2...*

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

See [\[Concept Index\]](#), page 112 for: Autozyg population model parameters.

9.8.7 Autozyg computational parameters

- See [Section 7.4 \[ibddrop statements\]](#), page 42, for statements specifying the proband gametes and locus window for `lm_auto`.
- See [Section 7.4 \[ibddrop statements\]](#), page 42, for the statement for setting the seeds for the LM-sampler.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`select trait K`

`lm_pval` needs to know which members of the pedigree are affected with the disease. Discrete or genotypic data for the selected trait is used to determine the disease affection status of the individuals. Here, `lm_pval` is to determine the affected individuals from the trait data in the pedigree file. A trait genotypic code of 3 (genotype (1 2) or (2 1)) or 4 (genotype (2 2)) indicates an affected individual. The trait number, *K*, determines the position of this genotypic code in the pedigree records (see [Section 2.7 \[Pedigree file description statements\]](#), page 12).

See [\[Concept Index\]](#), page 112 for: Autozyg computational parameters, marker data, missing marker data, scoreset, proband gametes, locus window, scoreset, window patterns, trait data, genotypic trait, discrete trait, quantitative trait, incomplete penetrance, affected individuals.

9.8.8 Autozyg MCMC parameters and options

All the statements described in see [Section 8.6 \[MCMC parameter statements\]](#), page 49 for specifying the MCMC parameters are used by the Autozyg programs.

Please see that section for details regarding:

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

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

check progress I MC iterations
```

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

```
set MCMC markers only
```

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

See [\[Concept Index\]](#), page 112 for: Autozyg MCMC parameters and options.

10 Estimating *ibd* Based Test Statistics by MCMC

See [Concept Index], page 112 for: *ibd*-based tests.

10.1 Introduction to `lm_ibdttests` and `civil`

See [References], page 110, for details of the cited papers.

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

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

A new ($lambda,p$) model has been implemented in `lm_ibdttests`. 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_ibdttests` only allows the *ibd* measure $T=Spairs$ in the delta model set-up. The program returns the p-values of the likelihood-ratio statistics under each of these two models. For a detailed description of the ($lambda,p$) and delta models, see [Bas10]. For a real data analysis using `lm_ibdttests`, see [Sie05].

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

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

`civil` computes marginal and conditional test p-values using Monte Carlo method: to approximate the null test statistic distributions, the program will hold trait values fixed and resample the inheritance vectors. The inheritance vectors along a chromosome should follow a Markov Chain distribution in genomic regions absent of causal genetic variants. In a marginal test, the null inheritance vectors are sampled from the marginal distribution

of the Markov Chain, which is uniform over the set of all possible inheritance vectors (see [Section 9.1 \[Introduction to `lm_auto` `gl_auto` and `lm_pval`\], page 51](#)). In a conditional inheritance vector test, the inheritance vectors are sampled from the conditional distribution the inheritance vector at the test position given the observed inheritance vectors at the two conditioning positions, as determined by the Markov Chain distribution.

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

See [\[Concept Index\], page 112](#) for: `lm_ibdtests` introduction, `civil` introduction, `vc-score` and `w-score`.

10.2 Sample `lm_ibdtests` parameter file

The example parameter file for `lm_ibdtests`, '`ped73_ibdt_IBD.par`', may be found in the '`TraitTests`' subdirectory of '`MORGAN_V30_Examples`'. Several lines in the example parameter file have been explained in previous sections of the tutorial, only the sections requiring additional explanation are shown below.

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

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

compute scores every 100 iterations
```

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

The '`compute ibd statistics`' statement must be included in the parameter file when running `lm_ibdtests`. The next line instructs the program to use Spairs and Srobdom to perform the *ibd* tests. The '`set ibd tests`' command calls for both normal and permutation tests to be run. The next line is needed since permutation test were requested in the previous line; it specifies how many permutations are to be used in the calculations. In this case, the

default (999) is specified; it is recommended that at least 50 permutations are used. The last line in the parameter file is used to specify when to compute scores, the default is every MCMC iteration.

See [Concept Index], page 112 for: sample parameter file for `lm_ibdttests`.

10.3 Sample `lm_ibdttests` output

Under the subdirectory ‘TraitTests’, run the example with the following command

```
./lm_ibdttests 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
*****
```

locus	pos(Haldane cM)		Spairs	Srobdom
	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
*****
```

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

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

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

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

See [Concept Index], page 112 for: `lm_ibdtests` sample output.

10.4 Sample civil parameter file

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

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

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

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

```
input pedigree file 'civil.ped'

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

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

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

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

```

input pedigree record trait 2 real 3

set trait 2 tloc 12

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

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

```

The two ‘map test tloc’ statements are required by MORGAN, but the numbers in those lines will not be used by *civil*. The values of ‘additive variance’ and ‘residual variance’ specified here will be used by *civil* only when ‘use_sample_variance’ is set to ‘no’ in the extra parameter file (see below). The ‘genotype means’ will be used only if ‘use_sample_mean’ is set to ‘no’ in the extra parameter file.

Additional informations about marginal and conditional test setup are provided to *civil* through an ‘extra file’.

```
input extra file 'civil.vc.xtra'
```

The outline of the extra file is as follows (for an example, see ‘Autozyg/Gold/civil.vc.xtra’):

```

## inheritance vector file name (.oscor file)
civil.oscor
## output file directory
.
## output file keyword
civil
## info on the oscore file ...
n_mcmc 10
order 0
## trait model parameters ...
pD 0.3
use_sample_mean yes
mu 0
use_sample_sd yes
## marginal test parameters
test_statistic vc
n_mc 9999
n_pos 101
test_pos 0 4 8 12 ...
## conditional test parameters
test_statistic vc
n_mc 999
n_pos 81
test_pos 40 44 48 ...

```

```

test_pos_l 0 4 8 ...
test_pos_r 80 84 88 ...

```

The first 6 lines provide the name of the `gl_auto` output file (line 2), the name of the output directory (line 4), and a keyword for naming the output files (line 6). `civil` will create four output files, suffixed by `*.miv.p.out`, `*.miv.t.out`, `*.civ.p.out`, and `*.civ.t.out`, in the output directory. The four files store marginal and conditional test statistic values and p-values.

The section following `## info on the oscore file ...` specifies the number of MCMC scans in the `gl_auto` output file and whether the output is arranged by component or not, with 1 meaning yes and 0 no. If the lines in the `sgl_auto` output is arranged by component, the lines will be rearranged so that they are ordered by MCMC scan and a new file will be created to store the rearranged output file.

The section following `## trait model parameters ...` specifies the rare allele frequency of the putative causal variant and specifies how to estimate mean trait value and residual standard error for the trait values: if `use_sample_mean yes`, then `civil` will use the raw sample mean to estimate the mean trait value, otherwise the mean value specified in the next line will be used. If `use_sample_sd yes`, then `civil` will use the sample sd to estimate residual standard error, otherwise residual standard error will be estimated by $\sqrt{\text{residual variance} + \text{additive variance}}$ using values provided in the main `civil` parameter file.

The section following `## marginal test parameters` specifies the test statistic, the number of Monte Carlo runs for simulating the null distribution (not to be confused with the count of MCMC realizations in the `gl_auto` output scores file), the number of tests requested and the indices to the test positions for the marginal tests. Currently, two test statistic options `vc` and `w` are available. In this example par file, we ask `civil` to perform 101 marginal tests at positions indexed by 0, 4, 8, ..., 404.

The section following `## conditional test parameters` specifies the test statistic, the number of Monte Carlo runs for simulating the null distribution, the number of tests requested, indices to the test positions, indices to the left and right conditioning positions (one line for each set of positions) for conditional tests. In this example par file, we ask `civil` to perform 81 condition tests. The first conditional test will be at position indexed by 40 and be conditioned on positions 0 and 80.

Note that the test positions have to be a subset of marker positions. The idea is to run `gl_auto` using a set of dense markers that should include all potential test and conditioning positions, although not necessarily all markers in the marker data file. When performing marginal and conditional tests, less dense marker positions can be used.

Currently, this extra file has rigid format requirement. Comment lines (starting with `##`) can be modified, but no line should be deleted or added, nor should existing lines be broken into multiple lines. The example xtra file `Gold/civil.vc.xtra` can be used as a template for creating new xtra file.

See [\[Concept Index\]](#), page 112 for: sample parameter file for `civil`, latent p-values, randomized p-values.

10.5 Sample civil output

Since `civil` is still a beta-test program, it does not have an example in the ‘MORGAN_V30_Examples’ directory. Instead, reference is made to the gold standard examples in the main MORGAN source directory, in the subdirectory ‘Autozyg/Gold’.

Before running the program `civil`, the user needs to run `gl_auto` to obtain an MCMC sample of whole chromosome realizations of meiosis indicators. See [Section 9.5 \[Running gl_auto example and sample output\]](#), page 59, for details. Under the directory ‘Autozyg/Gold’, the output file ‘`civil.oscor`’ from a previous `gl_auto` run is provided for demonstration and testing purpose.

Before running `civil`, an output subdirectory must exist. If `vc` is specified as the test statistic, create a subdirectory named ‘`vc`’ for storing temporary files in the user specified output file directory; if `w` is specified as the test statistic, create a subdirectory named ‘`w`’.

To run the example in ‘Autozyg/Gold’ make sure the following files are present there: `civil.vc.par`, `civil.vc.xtra`, `civil.ped`, `civil.markers`, `civil.oscor`.

In the ‘Autozyg/Gold’ directory, run `civil` by typing

```
../civil civil.vc.par > civil.vc.out
```

Information on the progress of the program will be printed to `stdout`, together with summary information about the pedigrees, markers, trait values, and marginal and conditional test setup. For a large number of pedigrees, `civil` can take several hours to finish. Once the program is finished, four output files, `*.miv.?.t.out`, `*.miv.?.p.out`, `*.civ.?.t.out`, `*.civ.?.p.out`, will be written to the specified output file directory: ‘`*`’ is the output file keyword specified in the `xtra` file and ‘`?`’ is the name of the specified test statistic (‘`w`’ or ‘`vc`’). They store marginal test statistic values, marginal test p-values, conditional test statistic values, conditional test p-values.

The upper left portion of a marginal test p-values file ‘Autozyg/Gold/civil.miv.m.p.out’ is shown below:

test_pos	test_map	pval0	pval1	pval2	...
0	0.000000	0.214400	0.098700	0.357800	...
4	1.000000	0.305700	0.108900	0.142800	...
8	2.000000	0.327400	0.133200	0.132700	...
...					

In this output file, the first row is the header. Each of the remaining rows corresponds to one marginal test. The first two columns are the index and the map position of the test position. The columns 3 to $N + 2$ are the test p-values, one for each MCMC realization of the meiosis indicators. The layout of the marginal test statistic file is similar.

The conditional test p-values file ‘Autozyg/Gold/civil.civ.m.p.out’ has more columns. For each test, the first 6 columns now correspond to indices to conditional test position, left conditioning position and right conditioning position; then map positions of the conditional test position, left conditioning position and right conditioning position. Starting from column 7 are the N p-values, one for each MCMC realization.

Many temporary files will also be created under the subdirectories ‘`vc`’ or ‘`w`’ of the output directory. These files store intermediate results for computing the test scores. These results will be reused to save time when more tests need to be performed: for example, the user

may want to perform more marginal and conditional tests at different test or conditioning positions.

However, if pedigree structures or trait values in the pedigree file, or trait parameters in the ‘extra file’ file have changed since last run, these temporary files should not be reused and should be deleted before running `civil`. If pedigree structures have changed, `gl_auto` also need to be rerun. Use the `overwrite` option for the `gl_auto` output scores file, to overwrite the previous file, and/or rename the previous file if you wish to retain it.

See [Concept Index], page 112 for: `civil` sample output.

10.6 `lm_ibdttests` and `civil` statements

The programs `lm_ibdttests` and `civil` use the pedigree, and genetic map and marker statements of previous sections.

- For the MCMC statements used by `lm_ibdttests` see Section 8.6 [MCMC parameter statements], page 49.
- For the `gl_auto` statements used by `civil` see Section 9.8 [Autozyg statements], page 61.

The following statements are specific to `lm_ibdttests`:

`compute (ibd | likelihood-ratio) statistics`

Required: one of the two options must be specified.

`output (sampler | permutation) seeds only`

The program `lm_ibdttests` uses random seeds for its permutation testing in addition to the usual MCMC sampler seeds. If an output seed file is named, both ending permutation and sampler seeds will be saved unless only one or the other is requested.

`set ibd measures [Spairs] [Srobdom] [Saffect] [Slambda]`

Optional. `lm_ibdttests` uses 1 to 4 measures to perform *ibd* tests for lineage; these are specified in the order [Spairs] [Srobdom] [Saffect] [Slambda]. Spairs, Srobdom, and Slambda may be specified for both normal and permutation tests; Saffect may not currently be specified with the normal tests option.

`set ibd tests [normal] [permutation]`

Optional. Normal and/or permutation tests may be specified.

`set ibd permutations I`

Optional. Need to be specified when the permutation test is requested through ‘set ibd tests’. The default is 999. It is recommended that at least 50 permutations are used.

`set likelihood-ratio lambda-p model gridpoints I1 I2`

When the `lambda_p` measure is used for the chi-square likelihood-ratio test), the number of gridpoints may be specified. The number `I1` is the number of gridpoints in the interval for the lambda-parameters of the model, and `I2` is the number of gridpoints in the interval for `p`. The default is 6 and 9, respectively.

`set likelihood-ratio measures [delta] [lambda_p]`

When computing the chi-square likelihood-ratio test, the choice of measures is `delta` and/or `lambda_p`, in the order [delta] [lambda_p]. The default is ‘delta’.

set likelihood-ratio tests

When computing likelihood-ratio statistics, chi-squared tests are performed. Thus, this statement is presently redundant, as there is no choice in tests.

set permutation seeds *H1 H2*

The program `lm_ibdtests` uses random seeds for its permutation testing in addition to the usual MCMC sampler seeds. The seeds may be specified in the 'input seed file' or in the parameter file: otherwise default seeds will be used.

The program `civil` has no program-specific parameter statements. Instead information is provided to `civil` using the **input extra file statement**:

input extra file *filename*

Required

For information about the contents of the extra file see [Section 10.4 \[Sample civil parameter file\]](#), page 70.

See [\[Concept Index\]](#), page 112 for: `lm_ibdtests` statements, `civil` statements, *ibd* measures, likelihood-ratio measures.

11 Estimating Location lod Scores by MCMC

See [\[Concept Index\]](#), page 112 for: location lod scores estimates.

11.1 Introduction to `lm_linkage`, `lm_bayes`, `lm_twoqt1`, and `gl_lods`

The programs `lm_linkage`, `lm_bayes`, and `lm_twoqt1` are referred to as ‘Lodscore’ programs. The program `lm_linkage` replaces the two programs `lm_markers` and `lm_multiple` of pre-2011 versions of MORGAN. As of 2011, the program `lm_twoqt1` remains a beta test version.

The Lodscore programs use MCMC to perform multipoint linkage analysis and trait mapping on large pedigrees where many individuals may be unobserved and exact computation is infeasible. The data are the genotypes of observed individuals in the pedigree at marker loci and discrete or continuous trait data. As with exact methods of computing lod scores, the genetic model is assumed known. The only unknown parameter is the location of the trait locus. Therefore, the user is required to specify the marker locations, trait and marker allele frequencies and penetrance function. Presently, users are limited in their choice of penetrance function, but this is under revision and will change in future releases of MORGAN.

`lm_linkage` is an implementation of the Lange-Sobel estimator, using either the single- or multiple-meiosis LM-sampler: See [Section 8.4 \[Single and multiple meiosis LM-samplers\]](#), page 46. The Lange-Sobel estimate works reasonably well in reasonable time, provided a good MCMC sampler is used, and provided the trait data do not have strong impact on the conditional distribution of meiosis indicators. The `lm_linkage` program samples only the meiosis indicators at marker loci, and only conditional on the marker data. Even when the trait inheritance information is strong, the method can produce quite accurate lod scores in the absence of linkage, but it can be inaccurate in estimating the strength of linkage signals. As well as producing the lod score, our current implementation provides a batch-means pointwise estimate of the Monte Carlo standard error of the lod-score estimate. `lm_linkage` can work with genotypic, discrete or quantitative traits.

`lm_linkage` combines the earlier programs `lm_markers` and `lm_multiple`. The original `lm_multiple` program and multiple-meiosis sampler are the work of Liping Tong [TT08]. As well as allowing use of either the single- or multiple-meiosis LM-sampler, the `lm_linkage` program optionally perform exact lodscore computations on small pedigree components, and includes better exact computation and pedigree peeling options for use in the lod score estimator (see [Section 8.3 \[Exact HMM computations\]](#), page 45).

`lm_bayes` is an alternative method implemented for genotypic or discrete traits. The MCMC performance is better than for the old `lm_markers` program, but it has other computational overheads. `lm_bayes` samples trait locations from a posterior distribution, and then divides it by the prior to produce the likelihood and hence the lod score. Estimation is in two phases. A preliminary run with discrete uniform prior gives order-of-magnitude relative likelihoods. Then, using the inverse of these likelihoods as prior weights of a ‘pseudo-prior’ distribution. Using this ‘pseudo-prior’ a second run is made to estimate the likelihood. The purpose of the ‘pseudo-prior’ is to produce an approximately uniform posterior, so that likelihoods will be well estimated at all test positions. It is important that the initial run is long

enough for all test positions to be sampled, and for the unlinked trait position to have a reasonable number of realizations. For locations at which lod scores are very negative, or for the unlinked position when there is some linked location with strong positive lod score, this can be problematic.

Our current implementation of `lm_bayes` provides two lod score estimates. The first is a crude estimate which counts realizations of locations sampled to estimate the posterior: as can be seen from the output this can be quite erratic. The Rao-Blackwellized estimator is much preferred, and produces good estimates in reasonable time. The `lm_bayes` program is the work of Andrew George [GT03,GWT05].

The beta-test program `lm_twoqtl` does parametric linkage analysis for a quantitative trait model having one or two linked QTL and a polygenic component. Each QTL is diallelic with 3 different genotypic means. The Normally distributed polygenic component does not include dominance, and the environmental contribution is has a Normal distribution with mean zero and uncorrelated among individuals. The program output consists of MCMC-based lod score estimates of the joint locations of the one or two contributing QTL. As of 2011, the program uses the same MCMC options as `lm_linkage` for sampling descent at marker loci conditional on marker data. Conditionally on these realizations the program then uses exact computation (on very small pedigrees) or an additional level of Monte Carlo to estimate the relevant lod score contributions. The original versions of the `lm_twoqtl` were the work on YunJu Sung [STW07,SW07].

The beta-test program `gl_lods` computes lod score contributions for a discrete or continuous trait given a set of `ibd_graphs` across the chromosome, produced by `gl_auto`: See [Section 9.1 \[Introduction to `lm_auto` `gl_auto` and `lm_pval`\], page 51](#). If the `gl_auto` run uses the ‘`set MCMC markers only`’ option, then the overall lod score computed by `gl_lods` is identical to that produces by `lm_linkage` when the same MCMC options are used in the in `gl_auto` and in `lm_linkage`. `gl_lods` uses the same parameter statements as `lm_linkage` ([Section 11.8 \[Location lod scores statements\], page 91](#)), but ignores some input statements and uses others in a non-standard way. For further information on the motivation for splitting of the `lm_linkage` lod score computation into the generation of marker-based `ibd_graphs` (using `gl_auto`) followed by trait-likelihood computation on the `ibd_graphs`: See [Section 11.6 \[Parameter files for the `gl.lods` program\], page 86](#). See also [Tho11].

See [\[References\], page 110](#), for details of the cited papers.

See [\[Concept Index\], page 112](#) for: Markov chain Monte Carlo, `lm_linkage` introduction, `lm_bayes` introduction, `lm_twoqtl` introduction, meiosis indicators, multiple meiosis sampler.

11.2 Sample parameter files for `lm_linkage` and `lm_bayes`

There are three example parameter files in the ‘`Lodscores`’ subdirectory: ‘`ped73_ge.par`’, ‘`ped73_ph.par`’ and ‘`ped73_qu.par`’. These files are examples of how to analyze genotypic, discrete (phenotypic), and quantitative (continuous) traits, respectively. Each of these files is written for use with `lm_linkage` since this is our preferred program and can analyze genotypic, discrete, and quantitative traits. The program `lm_bayes` will run with the same parameter files ‘`ped73_ge.par`’ and ‘`ped73_ph.par`’, but will adopt defaults for several statements specific to this program and will generate warning for others not implemented for

`lm_bayes`. If `lm_bayes` is run using `'ped73_qu.par'`, all statements regarding quantitative traits will be ignored, and the program will use default genotypic data.

The marker and MCMC information is very similar for all three parameter files. For `'ped73_qu.par'` it is as follows:

```

set printlevel 5

input pedigree file '../ped73.ped'
input marker data file '../ped73.marker.missing'
input seed file      '../sampler.seed'
output overwrite seed file      '../sampler.seed'

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

select all markers
select trait 1
set trait 1 tloc 1
map test tloc 1 all interval proportions 0.3 0.7
map test tloc 1 external recomb fracts 0.05 0.15 0.3 0.4 0.45

sample by scan
set L-sampler probability 0.2

set burn-in iterations 150
set MC iterations 3000
check progress MC iterations 1000

set global MCMC
use single meiosis sampler

```

The pedigree file specified by the `'input pedigree file'` statement can contain multiple traits. As discussed in previous sections, the marker map, allele frequencies and genotypes can be contained in the parameter file or in a separate file specified by the `'input marker data file'` statement as in the example above.

As in other programs, the trait data are included in the pedigree file. The `'select trait'` statement tells the program which trait in this file is to be analyzed, and the `'input pedigree record trait'` indicates where the data are to be found, while the `'set trait ...tloc...'` statement connects the trait with a specific tloc for this analysis.

The two `'map test tloc'` statements give trait locus test positions at which the lod scores should be calculated. When the trait locus is located between two markers, the position is specified in terms of the proportional genetic distance between the two markers (this option makes handling gender-specific maps easy). In this example, the test trait positions are specified to be at 30 and 70 percent of the interval. The second `'map test tloc'` statement allows test trait locus positions located before the first marker or after the last marker to be specified; the positions are specified explicitly in terms of recombination fractions (or genetic distances) with the nearest marker locus. Note that an external recombination

fraction of 0.5 is not necessary since the likelihood of an unlinked trait locus is always used as a reference when computing the lod scores.

The final seven statements give MCMC specifications. The ‘`sample by scan`’ statement instructs the program to update all the meiosis indicators, **S**, at each iteration, in an order determined by random permutation. The alternative ‘`sample by step`’ updates only one locus (L-sampler) or only one meiosis (M-sampler) in each iteration. The ‘`sample by scan`’ statement is the default and strongly recommended. The L-sampler probability is set at 20 percent, which is often a good choice. For a detailed discussion of effects of varying L- to M-sampler ratio, see section 10.6 in [Tho00].

In the ‘`set burn-in iterations`’ statement, 150 burn-in iterations, are requested. The next statement requests 3000 MCMC iterations; for each realized set of marker-location inheritance vectors the trait-likelihood contribution will be computed at each test position of the trait locus. This is for demonstration purposes only. For real data analyses, use longer runs, on the order of 10^5 MCMC iterations. The last statement in this group tells the program to report progress every 1000 iterations.

Although the `lm_linkage` program can use the multiple-meiosis sampler, and this is recommended, the final two statements here specify ‘`set global MCMC`’ and ‘`use single meiosis sampler`’. Thus, for this example, the single-meiosis sampler will be used (as in the old `lm_markers` program) and MCMC will be performed globally over all pedigree components, rather than component-by-component. This provides an example of how these options may be used for compatibility with older examples.

For more details of the MCMC specifications see [Section 8.6 \[MCMC parameter statements\]](#), [page 49](#).

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 2.7 \[Pedigree file description statements\]](#), [page 12](#)). The three trait data types discussed in this example are implemented by including the following statements in the parameter file discussed above. Note the trait and numbers are arbitrary, but the connection must be made consistently through the file.

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

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

select trait 3
set trait 3 tloc 1
set tloc 1 allele freqs 0.4 0.6
```

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

```
set trait 2 data discrete
input pedigree record trait 2 integer 4

select trait 2
set traits 2 tlocs 1
set traits 2 for tlocs 1 incomplete penetrance 0.05 0.6 0.95
```

```
set tlocs 1 allele freqs 0.4 0.6
```

Recall that for discrete data, one must specify the penetrances (see [Section 9.8.7 \[Autozyg computational parameters\]](#), page 64).

‘ped73_qu.par’ specifies a quantitative trait with the following statements:

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

select trait 1
set trait 1 tloc 1
set trait 1 for tlocs 1 genotype mean 90.0 100.0 110.0
set trait 1 residual variance 25.0
set tloc 1 allele freqs 0.4 0.6
```

When using a quantitative trait, genotypic means and residual variance must be specified. Additive variance can be specified with the statement ‘set trait ... additive variance’. The default value is zero.

The ‘set tloc ... allele freqs’ statement specifies allele frequencies at the trait locus. If the allele frequencies sum to less than 1, a warning message will be issued:

```
Sum of allele frequencies is not in range .9999, 1.0001 (W)
```

If the allele frequencies sum to above 1.0001, the program quits and generates an error message.

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

	Genotypic	Phenotypic	Quantitative
	<i>ped73_ge.par</i>	<i>ped73_ph.par</i>	<i>ped73_qu.par</i>
lm_linkage	Yes	Yes	Yes
lm_bayes	Yes	Yes	No

See [\[Concept Index\]](#), page 112 for: sample parameter file for **lm_linkage**, sample parameter file for **lm_bayes**, gender-specific maps, meiosis indicators, L-sampler, M-sampler, multiple meiosis sampler, genotypic trait specification for lod score calculation, phenotypic trait specification for lod score calculation, discrete trait specification for lod score calculation, quantitative trait specification for lod score calculation, continuous trait specification for lod score calculation.

11.3 Running lm_linkage examples and sample output

lm_linkage can be run with all three parameter files in the ‘Lodscores/’ subdirectory. As usual, the syntax for running the program is:

```
./lm_linkage <parameter file>
```

This section describes the output obtained by using the parameter file ‘ped73_qu.par’. To run the example, type:

```
./lm_linkage ped73_qu.par
```

The interesting part of the output is the LodScore estimates. For each test position, we have the estimated lod score and the estimated Monte Carlo standard error.

```
LodScore estimates by Rao-Blackwellized computation:
```

Trait pos # or marker	position (Haldane cM)		LodScore	StdErr
	male	female		
1	-115.129	-115.129	0.0303	0.0005
2	-80.472	-80.472	0.0558	0.0012
3	-45.815	-45.815	0.0779	0.0031
4	-17.834	-17.834	-0.0306	0.0080
5	-5.268	-5.268	-0.2811	0.0142
marker-1	0.000	0.000	-0.4986	0.0195
6	3.000	3.000	-0.4469	0.0141
7	7.000	7.000	-0.4342	0.0230
marker-2	10.000	10.000	-0.4605	0.0363
8	13.000	13.000	-0.4254	0.0247
9	17.000	17.000	-0.4454	0.0209
marker-3	20.000	20.000	-0.5301	0.0197
10	23.000	23.000	-0.3174	0.0211
11	27.000	27.000	-0.1176	0.0233
marker-4	30.000	30.000	-0.0052	0.0259
12	33.000	33.000	0.5058	0.0208
13	37.000	37.000	0.8794	0.0159
marker-5	40.000	40.000	1.0772	0.0138
14	43.000	43.000	0.9832	0.0156
15	47.000	47.000	0.8432	0.0213
marker-6	50.000	50.000	0.7210	0.0252
16	53.000	53.000	0.6558	0.0256
17	57.000	57.000	0.5140	0.0271
marker-7	60.000	60.000	0.3522	0.0288
18	63.000	63.000	0.0113	0.0225
19	67.000	67.000	-0.5473	0.0123
marker-8	70.000	70.000	-0.9543	0.0095
20	73.000	73.000	-0.4578	0.0212
21	77.000	77.000	-0.1866	0.0178
marker-9	80.000	80.000	-0.1135	0.0116
22	83.000	83.000	0.0888	0.0091
23	87.000	87.000	0.3132	0.0064
marker-10	90.000	90.000	0.4544	0.0071
24	95.268	95.268	0.6010	0.0046
25	107.834	107.834	0.6423	0.0028
26	135.815	135.815	0.4017	0.0011
27	170.472	170.472	0.1762	0.0003
28	205.129	205.129	0.0758	0.0001

For more information regarding the MCMC parameters and diagnostic output, See [Section 8.5 \[MCMC computational options\]](#), page 47.

See [\[Concept Index\]](#), page 112 for: running `lm_linkage` examples, `lm_linkage` sample output.

11.4 Running `lm_bayes` examples and sample output

Under the subdirectory ‘Lodscores/’, run the `lm_bayes` example on the discrete (phenotypic) trait data by typing:

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

The results from `lm_bayes` are the lod scores toward the end of the output. Two estimates of the lod scores are provided: (1) count realizations of locations sampled to estimate the posterior probability (‘crude’) and (2) Rao-Blackwellized estimator (‘R-B’). Both are provided for comparison, but the latter should be more accurate.

LodScore estimates:

Trait or marker	pos # marker	position (Haldane cM)		pseudo prior	freq visited	LodScore	
		male	female			crude	R-B
	0	unlinked	unlinked	0.025023	94	NA	NA
	1	-115.129	-115.129	0.025276	66	-0.1580	-0.0046
	2	-80.472	-80.472	0.025727	77	-0.0987	-0.0125
	3	-45.815	-45.815	0.027843	96	-0.0372	-0.0473
	4	-17.834	-17.834	0.037973	71	-0.3030	-0.1825
	5	-5.268	-5.268	0.057289	96	-0.3506	-0.3583
marker-1		0.000	0.000	NA	NA	NA	NA
	6	3.000	3.000	0.078826	89	-0.5221	-0.4919
	7	7.000	7.000	0.086379	88	-0.5667	-0.5255
marker-2		10.000	10.000	NA	NA	NA	NA
	8	13.000	13.000	0.092502	87	-0.6014	-0.5456
	9	17.000	17.000	0.090858	94	-0.5600	-0.5386
marker-3		20.000	20.000	NA	NA	NA	NA
	10	23.000	23.000	0.063483	109	-0.3400	-0.3738
	11	27.000	27.000	0.044111	103	-0.2065	-0.2086
marker-4		30.000	30.000	NA	NA	NA	NA
	12	33.000	33.000	0.026053	114	0.0663	0.0203
	13	37.000	37.000	0.018403	103	0.1731	0.1698
marker-5		40.000	40.000	NA	NA	NA	NA
	14	43.000	43.000	0.011818	100	0.3527	0.3585
	15	47.000	47.000	0.009347	90	0.4088	0.4600
marker-6		50.000	50.000	NA	NA	NA	NA
	16	53.000	53.000	0.010351	121	0.4930	0.4236
	17	57.000	57.000	0.014614	121	0.3432	0.2804
marker-7		60.000	60.000	NA	NA	NA	NA
	18	63.000	63.000	0.023348	96	0.0392	0.0769
	19	67.000	67.000	0.030506	123	0.0307	-0.0412
marker-8		70.000	70.000	NA	NA	NA	NA
	20	73.000	73.000	0.033357	136	0.0356	-0.0903
	21	77.000	77.000	0.030400	124	0.0358	-0.0514
marker-9		80.000	80.000	NA	NA	NA	NA
	22	83.000	83.000	0.024811	96	0.0128	0.0282
	23	87.000	87.000	0.019535	144	0.2928	0.1160

marker-10	90.000	90.000	NA	NA	NA	NA
24	95.268	95.268	0.013755	110	0.3281	0.2561
25	107.834	107.834	0.013714	125	0.3849	0.2600
26	135.815	135.815	0.018372	132	0.2816	0.1339
27	170.472	170.472	0.022361	108	0.1091	0.0489
28	205.129	205.129	0.023966	87	-0.0149	0.0188

Note that `lm_bayes` does not provide lod scores at the marker locations.

See [\[Concept Index\]](#), page 112 for: running `lm_bayes` examples, `lm_bayes` sample output, Rao-Blackwellized estimates.

11.5 Running `lm_twoqtl` examples and sample output

The program `lm_twoqtl` remains beta test, so that instead of examples in ‘MORGAN_V30_Examples’ we describe the gold standard example in the main MORGAN source directory in subdirectory ‘Lodscore/Gold’. However, much work has been done on `lm_twoqtl`, so that its marker-based MCMC is now as for `lm_linkage`. Four gold-standard examples of the `lm_twoqtl` parameter files and output may be found in the ‘Lodscore/Gold’ directory of MORGAN. Examples 1 and 3 are for a single trait locus, and 2 and 4 use two QTL. Examples 1 and 2 use exact computation for the trait lod score contributions on these very small examples. Examples 3 and 4 use Monte Carlo. We will use example 4 in this tutorial description, since this is the most general and novel. To create other examples, copy one of these files and replace the parameters in the file with those that you want to specify.

The various trait-model options for `lm_twoqtl` are summarized in the following table:

Additive Genetic Variance:	Zero	Positive
Number of QTL: 1	one locus	one locus plus polygene
2	two loci	two loci plus polygene

Trait models can be any of the above four entries. However, for a one-locus trait model with no polygenic component, the program `lm_linkage` will provide more accurate results more quickly.

The lod score is estimated on a one-dimensional grid of points for one QTL, and a two-dimensional grid of points for two QTL. In the future the new parameter statement

```
map [chromosome I] test tlocs L1 L2 jointly at markers J11 J12 ...
```

will allow two-locus lod score programs that provide lod scores at arbitrary pairs of marker positions.

The content of file ‘twoqtl4.par’ (reordered slightly for clarity) is:

```
use single meiosis sampler          # Select the MCMC sampler to be used.

set printlevel 5                    # Include everything in the output file.

set sampler seeds 0x53f78285 0xdfbca001
set trait seeds 0x53f78285 0xdfbca001

input marker data file './twoqtl.markers'
input pedigree file './twoqtl.ped'
```

```

output extra file './twoqtl_batch4'

select all markers
select trait 1

set trait 1 multiple tlocs 1 2
set tloc 1 allele freqs 0.1 0.9
set tloc 2 allele freqs 0.3 0.7

# requests for grid of tloc positions for lod scores
map test tloc 1 all interval proportions 0.5
map test tloc 1 external recomb fracts 0.3
map test tloc 2 all interval proportions 0.5
map test tloc 2 no default external positions

# standard MCMC requests
use sequential imputation for setup
use 100 sequential imputation realizations for setup
sample by scan
set L-sampler probability 0.2
set burn-in iterations 10
set MC iterations 60
compute scores every 10 iterations

# lodscore scoring requests
set 3 batches MC variance estimation
check progress 20 MC iterations
use MC summation for trait
use 5 MC realizations for trait
use multiplier 1 MC realization for null

# quantitative trait model specification
set trait 1 data quantitative
set trait 1 for tloc 1 genotype mean -2.0 0.0 2.0
set trait 1 for tloc 2 genotype mean -3.0 0.0 3.0
set trait 1 residual variance 1.0
set trait 1 additive variance 1.0

```

Note the number of MCMC scans (60) is very small, as also are the number of Monte Carlo realizations to be used in evaluating the trait likelihood contributions (5). Additionally, only every 10th MCMC scan is used for computing lod score contributions. This is reasonable, in that for `lm_twoqtl` lod score computation is computationally intensive, so that the standard procedure of scoring every scan is not efficient. However, with only 60 total scans, this means lod scores are based on only 6 realizations of inheritance conditional on the marker data. The example is for illustrative purposes only; in real examples much more Monte Carlo

would be required both in the marker-based MCMC and for estimating trait contributions to each score.

Most statements are as for earlier lod-score programs and can be found in the [Statement Index], page 116. The statements included in this example that require additional comment are

```
use single meiosis sampler
```

The old single-meiosis sampler is specified for consistency with earlier results; in practice the `multiple meiosis sampler` is preferred.

```
set trait 1 multiple tlocs 1 2
```

This statement specifies that trait 1 is contributed to by both tloc 1 and tloc 2.

```
set trait 1 for tloc 1 genotype mean -2.0 0.0 2.0
```

```
set trait 1 for tloc 2 genotype mean -3.0 0.0 3.0
```

The genotypic means for tlocs 1 and 2 are set separately, which will imply their additive contribution to trait 1. If the tlocs are not to contribute additively, the user should instead use the statement

```
set trait 1 for tlocs 1 2 joint genotype means ...
```

followed by 9 genotype means for the two tloc genotype combinations.

```
output extra file './twoqtl_batch4'
```

If an 'extra file' is specified, it is used by `lm_twoqtl` for output of batched means used in variance estimation. Most users will not require this file, although it can be used in MCMC diagnostics.

```
set 3 batches MC variance estimation
```

In this minimal example, the 6 scored realizations are divided into 3 batches, each of size 2. Again, real examples would use much larger number of realizations, and likely the default number of batches (20).

```
use MC summation for trait
```

```
use 5 MC realizations for trait
```

In this example, Monte Carlo summation is to be used for evaluating each trait-locus likelihood contributions conditional on marker-based realizations of inheritance, and for each such realizations there will be 5 realizations of trait allele descent.

```
use multiplier 1 MC realization for null
```

In this example the same number of realizations will be used to evaluate the marginal probability of trait data as are used for each lodscore grid point. In real examples, it may be advisable to increase this ratio, to obtain an accurate base level for the lodscore estimate.

The default procedure of estimation of lod scores on each of the two components separately is used. These are then summed, giving the final concluding output for this example:

```
# Lod score estimates and MC sd for entire pedigree:
```

```
# Index      TLoc1      TLoc2      LodScore      StdErr
```

1	-45.815	-45.815	1.4058	0.7605
2	-45.815	0.000	0.3872	0.6212
3	-45.815	10.000	0.4379	0.6232
4	-45.815	20.000	0.6616	0.6856
5	-45.815	65.815	0.4661	0.5916
6	0.000	-45.815	0.3503	0.6529
7	0.000	0.000	0.1343	0.6034
8	0.000	10.000	-0.0129	0.5600
9	0.000	20.000	1.0364	0.5772
10	0.000	65.815	1.1077	0.7724
11	10.000	-45.815	0.0983	0.6902
12	10.000	0.000	0.0461	0.5605
13	10.000	10.000	0.8585	0.6088
14	10.000	20.000	1.2174	0.5866
15	10.000	65.815	0.3512	0.7539
16	20.000	-45.815	1.5180	0.6479
17	20.000	0.000	0.7387	0.6567
18	20.000	10.000	0.9330	0.7020
19	20.000	20.000	1.2473	0.6162
20	20.000	65.815	1.3465	0.7664
21	65.815	-45.815	-0.3066	0.7764
22	65.815	0.000	-0.1259	0.6895
23	65.815	10.000	0.5714	0.7124
24	65.815	20.000	1.3537	0.6550
25	65.815	65.815	0.5343	0.7140

These results consist of base-10 lodscore estimates with MCMC standard deviations, estimated at the requested grid of test positions.

See [\[Concept Index\]](#), [page 112](#) for: running `lm_twoqt1` examples, `lm_twoqt1` sample output, map test tlocs jointly at markers.

11.6 Parameter files for the `gl_lods` program

The beta-test program `gl_lods` computes lod score contributions for a discrete or continuous trait given a set of `ibd_graphs` across the chromosome, produced by `gl_auto`: See [Section 9.1 \[Introduction to `lm_auto`, `gl_auto` and `lm_pval`\]](#), [page 51](#). If the `gl_auto` run uses the ‘set MCMC markers only’ option, then the overall lod score computed by `gl_lods` is identical to that produced by `lm_linkage` when the same MCMC options are used in the in `gl_auto` and in `lm_linkage`. `gl_lods` uses the same parameter statements as `lm_linkage` ([Section 11.8 \[Location lod scores statements\]](#), [page 91](#)), but ignores some input statements and uses others in a non-standard way.

Basically the goal of using `gl_auto` and `gl_lods` is to separate the lod score computation from the marker-based MCMC that produces realizations of the inheritance vectors at loci across the chromosome. The input to `gl_lods` consists of these realizations, in the format of `gl_auto` output compressed `ibd_graphs`, a specification of a trait model, and a list of individuals with their trait data. The additional information provided in the `gl_lods`

parameter files is to insure compatibility with other MORGAN programs, specifically `lm_linkage`. Some `lm_linkage` parameter statements are used by `gl_lods` in a non-standard way. Others are dummy statements.

The separation of lod-score computation and marker-based MCMC has several advantages:

- Lod score contributions are computed for each ibd graph, providing a distribution ("fuzzy lod") over the ibd-graph realizations.
- Use of IBD_Graph software to identify equivalent ibd graphs across realizations and across markers, will enable lod score contributions to be computed once only for each distinct ibd graph; (THIS IS NOT YET IMPLEMENTED WITHIN THE `gl_lods` PROGRAM)
- Lod scores for many different genetic models and for different traits may be computed on the same set of ibd graphs.
- The pedigree and marker information may be separated from the trait data. `gl_lods` uses no marker data or pedigree data, using instead the already generated ibd graph output of `gl_auto`. While a 'pedigree file' is included to satisfy MORGAN requirements, and to provide the trait data, the pedigree structure information is not used by `gl_lods` and can be completely 'dummy'; an example is given below.

As for other beta-test programs, we describe here the details of the gold standard example in the 'Lodscore/Gold' subdirectory of MORGAN. The files mentioned below are in that directory and the gold standard may be run from that directory as 'make gold.6'.

The following shows the general section of the parameter file 'ped47_gl_lods.par':

```
set printlevel 5           # Include everything in the output file.

# The MORGAN pedigree file provides the individual trait data but is
#   otherwise "dummy" Likewise a few dummy marker statements are required.

input pedigree file 'ped47_dummy.ped'
input marker data file 'ped47_dummy.markers'
input pedigree size 42    # (at least) the number in the dummy pedigree file
                        # In fact, there are 35 individuals in the pedigree file

# The extra input file contains the ibd graphs output by gl_auto.
# The number of individuals in these ibd graphs is determined by the program.
# The output scores file is a reduced version containing only individuals
#   who are found in the pedigree file, and who have data for the trait.

input extra file 'ped47_fgl.oscor'
output scores file 'ped47_fgl.reduce'

# Number of MC iterations provides the number of replicates in gl_auto file.
# If larger than the number in the file, a warning is given.
# If less than the number in the file, only this number will be used.
set MC iterations 1000

# The "select markers" statement will determine the markers at which
```

```

# lod scores are computed -- the "map test tloc" statement below is DUMMY.
select markers 2 4 5 8 10

# The following are required DUMMY statements.

select trait 1
set trait 1 tloc 11
set tloc 11 allele freqs 0.5 0.5

use single meiosis sampler      # DUMMY statement; there is no MCMC
map test tloc 11 at marker 1    # DUMMY statement; lm_linkage requires 1
                                # map test tloc statement
set L-sampler probability 0.5    # DUMMY statement; there is no MCMC

```

The pedigree is 'ped47_dummy.ped'. Note that the actual pedigree structure used in this file is not used, other than to establish to MORGAN that a single pedigree is involved. In reality, there may be several pedigree components, if ibd graphs on these components were generated jointly in the gl_auto output file.

```

# This pedigree file contains a subset of the 47 individuals in the gl_auto
# output file, and maybe other individuals.
# The pedigree file must include all those whose trait data are
# to be included:
# Individuals not in the gl_auto ibd graphs will be dropped.
# Individuals not observed for specified trait will be dropped.
# Individuals in the gl_auto file but not the pedigree file will be
# assumed unobserved (and dropped).

input pedigree record names 3 integers 7 reals 1

# The first three items are "names" which are character strings.
# They are the unique IDs of each individual and his/her parents.
# This pedigree file is DUMMY: the first two individuals are
# designated "dad" (male) and "mom" (female) and all others are
# specified as offspring of these two.
# The purpose of this structure is to specify a single pedigree component.
# If preferred the true original pedigree may be used.
#
# There follow 7 integers
# The first of these (4 th. item) is dummy gender (0,for the"kids")
# The next is an "observed" indicator: not used by gl_lods.
# The next is a trait genotype: (not used in these examples)
# The next is a trait phenotype:
# 0 is unobserved. 1 is unaffected, 2 is affected
# The next 2 code trait-locus inheritance patterns,
# not used by the gl_lods program.

```

```

# The final is a dummy trait indicating data for the examples,
# but not used by the gl_lods progrsm.
# Finally there is one real (read as double): this is a quantitative trait.
# Numbers with integer part 999 code for unobserved.
#
# Many individuals with no data are dropped: file has only 34 individuals
*****
302 0 0      1 1 3 2  1  1 1 105.945
306 0 0      2 1 3 1  1  1 1 99.822
307 302 306   0 1 4 2  1  1 1 111.696
308 302 306   0 0 0 0  1  1 0 999.5
3010 302 306  0 0 0 0 -1 -1 0 999.5
404 302 306   0 1 3 1  1  0 1 89.535
406 302 306   0 1 4 2  1  0 1 112.197
407 302 306   0 1 4 2  1  1 1 111.608
408 302 306   0 1 4 2  0  0 1 107.467
3050 302 306  0 0 0 0 -1 -1 0 999.5
410 302 306   0 1 4 1  1  1 1 92.77
411 302 306   0 1 3 2  1  1 1 106.814
412 302 306   0 1 4 1  1  0 1 99.992
3080 302 306  0 0 0 0 -1 -1 0 999.5
414 302 306   0 1 3 2  0  1 1 102.505
415 302 306   0 1 3 1  0  1 1 99.415
416 302 306   0 1 3 2  1  1 1 100.155
505 302 306   0 1 4 2  0  0 1 111.798
506 302 306   0 1 1 1  0  0 1 88.576
507 302 306   0 1 4 2  0  1 1 105.454
508 302 306   0 1 4 2  1  1 1 112.171
4050 302 306  0 0 0 0 -1 -1 0 999.5
509 302 306   0 1 3 1  0  0 1 99.518
510 302 306   0 1 3 1  1  1 1 98.543
511 302 306   0 1 3 2  1  1 1 111.349
512 302 306   0 1 3 2  1  0 1 100.304
513 302 306   0 1 4 2  1  0 1 103.615
514 302 306   0 1 4 2  0  0 1 115.385
515 302 306   0 0 0 0  0  0 0 999.5
516 302 306   0 1 4 2  0  0 1 111.138
5080 302 306  0 0 0 0 -1 -1 0 999.5
601 302 306   0 1 4 2  1  1 1 112.285
5160 302 306  0 1 3 1 -1 -1 0 97.043
5150 302 306  0 1 3 1 -1 -1 1 97.043
602 302 306   0 1 4 2  0  1 1 105.991

```

To satisfy MORGAN statements a few **dummy** marker statements are also required. The (dummy) marker data file is 'ped47_dummy.markers':

```
# THESE ARE TOTALLY DUMMY STATEMENTS
```

```

map marker dist  1 1 1 1 1 1 1 1 1 1          # dummy marker map
set markers 1 2 3 4 5 6 7 8 9 10 allele freqs 0.5 0.5 # dummy marker model
set markers 10 data
302 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2          # dummy marker data

```

Finally we require the `gl_auto` output scores file, 'ped47_fgl.oscor'. This file contains 100 ibd graphs on 47 individuals:

```

101, 102, 201, 202, 2010, 301, 302, 304, 2020, 305, 306, 307, 308,
3010, 404, 3040, 405, 406, 407, 408, 3050, 410, 411, 412, 3080, 414,
415, 416, 4040, 505, 506, 507, 508, 4050, 509, 510, 511, 512, 4080,
513, 514, 515, 516, 5080, 601, 5150, 602.

```

Comparing with the 35-member pedigree file we see:

- 5160 is in the pedigree file, not in the output scores file; will be dropped;
- about 11 individuals are in the above list, but not the pedigree file; these will be assumed unobserved, and dropped.
- another 7 individuals are in the pedigree file and the above list, but the trait data indicates them as unobserved; they will be dropped.

There will remain 27 individuals who will be in the reduced ibd graph file created by `gl_lods`.

Finally we require a trait-model specification; lod scores are computed under this model. The example file 'ped47_D.par' provides a provides the model for a discrete trait:

```

set trait 1 data discrete
input pedigree record trait 1 integer 4
set trait 1 for tloc 11 incomplete penetrances 0.1 0.6 0.9

```

Alternatively, the example file 'ped46_Q.par' provides the models for a quantitative trait:

```

set trait 1 data quantitative
input pedigree record trait 1 real 1
set trait 1 for tloc 11 genotype means 90.0 100.0 110.0
set trait 1 residual variance 25.0

```

11.7 Running `gl_lods` examples and sample output

There are two gold standards; one for a quantitative trait and one for a discrete trait, with gold-standard output files 'ped37_gl_lods_D.gold' and 'ped37_gl_lods_D.gold'. We describe here the output file 'ped37_gl_lods_D.gold'.

Much of the early output is standard MORGAN processing of the mainly dummy statements and can be ignored. Apart from a summary of the discrete trait phenotypes, the first output of interest occurs around line 145:

First it summarizes the input pedigree and ibd-graph files:

```

5 Selected markers:  2  4  5  8 10
nFGL from dummy pedigree input = 4
Opened input extra file "ped47_fgl.oscor"

```

```
Number of individuals in dgl-graph file is 47
Opened input extra file "ped47_fgl.oscor"
```

Then it process the lists, and notes the extra individual 5160 (see above)

```
Observed individual 5160 is not in DGL file:
Trait data on 5160 will be ignored (W)
Opened input extra file "ped47_fgl.oscor"
Opened output score file "ped47_fgl.reduce"
```

Then it processes its reduced list of observed individuals, reducing the ibd graphs, now of 27 individuals only. It also find the max number of FGL it will need: 26 in this example. It allocates for these; and provides also ‘NLocs’ which is one greater than the number of selected markers.

```
1000 Graphs were requested, but only 100 were given (W)
Number of individuals in each DGL graph = 27
Number of individuals in nghd structure = 35
Reset number of FGL = 26
Reopened reduced DGL graph file ped47_fgl.reduce
have alloced gen_pen nFGL=26, NLocs=6
```

Then the program produces the log-likelihood contributions at each of the 5 marker locations for each of the 100 reduced ibd graphs.

For the quantitative trait data and model, the gold-standard output file is ‘ped47_g1_lods_Q.gold’. The format of this file is identical to that for the discrete trait, except that now the data and trait model are for a quantitative trait.

11.8 Location lod scores statements

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

See [Concept Index], page 112 for: location lod scores statements, `lm_lods` statements, `lm_linkage` statements, `lm_bayes` statements.

11.8.1 Location lod scores computing requests

- For the ‘select’ statement for your MCMC simulation, See Section 9.8.1 [Autozyg computing requests], page 61. Select all or some of the markers and ‘trait 1’, and map this trait to a ‘tloc 1’ (this is the trait locus to be assigned varying test positions).

See [Concept Index], page 112 for: location lod scores computing requests.

11.8.2 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 9.8.2 [Autozyg file identification statements], page 62).

One additional statement is optional for `lm_bayes`:

```
output Rao-Blackwellized estimates file
```

If this file is specified, the set of Rao-Blackwellized lod score estimates at each trait position is written at the frequency specified in the ‘compute scores’ statement.

The same standard out file specifications are available for `lm_twoqt1`. In particular:

`output extra file`

This statement is used by `lm_twoqt1` to output batch mean estimates used in computing the estimated Monte Carlo standard error.

See [Concept Index], page 112 for: location lod scores file identification statements, Rao-Blackwellized estimates.

11.8.3 Location lod scores pedigree file description

All Lodscore programs use the general MORGAN pedigree file description statements (see Section 2.7 [Pedigree file description statements], page 12). One additional statement is optional for `lm_linkage`.

`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 2.7 [Pedigree file description statements], page 12) when the trait is quantitative, rather than discrete.

See [Concept Index], page 112 for: location lod scores pedigree file description, quantitative trait.

11.8.4 Location lod scores output file description

All Lodscore programs use the Autozyg output file description statements; See Section 9.8.4 [Autozyg output file description], page 63.

See [Concept Index], page 112 for: location lod scores output file description.

11.8.5 Location lod scores mapping model parameters

- See Section 5.4.2 [genedrop mapping model parameters], page 26, for statements specifying the genetic map for the markers.

The following statements describe the hypothesized trait locus (tloc) positions which are to be ‘tested’. That is, these are the positions at which lod scores will be computed.

`map [chromosome I] [gender (M | F)] test tloc L1 all interval proportions X1 X2 ...`

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

`map [chromosome I] [gender (M | F)] test tloc L1 intervals J1 ... proportions X1 X2 ...`

This statement specifies interval proportions, but between specific pairs of markers. Interval 1 is between markers 1 and 2, interval 2 is between markers 2 and 3, etc.

`map [chromosome I] [gender (M | F)] test tloc L1 (beginning | ending | external) ([Kosambi] distances | recombination fractions) X1 X2 ...`

This statement specifies trial trait positions on the chromosome before the first marker and/or after the last marker.

`map test tlocs L1 ... no default [interval | external] proportions`

This statement `map` to used to eliminate computation of lod scores at default interval and/or external positions on the active chromosome.a

`map [chromosome I] test tloc L at markers J1 ...`

This statement (new with MORGAN 3.0) is increasingly used with denser SNP marker data. If used, lod scores will be computed only at the positions of the specific markers. Note the marker indexing is by the count in the marker data file, not by selected marker.

`map [chromosome I] test tlocs L1 L2 jointly at markers J11 J12 ...`

This statement (not yet implemented) will allow two-locus lod score programs such as `lm_twoqt1` to compute lod scores only at any specified combination of marker positions rather than, as currently, on a grid.

See [Concept Index], page 112 for: location lod scores mapping model parameters, trait test positions, Kosambi map function, Haldane map function.

11.8.6 Location lod scores population model parameters

- See Section 5.4.3 [genedrop population model parameters], page 27, for statements specifying the allele frequencies for the markers and trait loci, and See Section 9.8.6 [Autozyg population model parameters], page 64, for statements specifying marker names.

See [Concept Index], page 112 for: location lod scores population model parameters.

11.8.7 Location lod scores computational parameters

- See Section 7.4 [ibddrop statements], page 42, for setting the sampler seeds.
- See Section 9.8.7 [Autozyg computational parameters], page 64, for specifying the marker data.
- See Section 9.8.7 [Autozyg computational parameters], page 64, for specifying the trait data as genotypic, quantitative or discrete and for specifying penetrances when trait data are discrete.
- See Section 5.4.4 [genedrop computational parameters], page 28, for setting genotype means for each tloc in the case of a quantitative trait.

The following additional statements are specific to lod score computations:

`set pseudo-priors X1 X2 ...`

This statement is optional for `lm_bayes`. The number of pseudo-priors is the number of test trait locus positions plus one. The first pseudo-prior is for the unlinked position; this should be assigned a positive value. All other pseudo-priors must be positive or zero. The set of pseudo-priors need not be normalized.

`set I batches MC variance estimation`

This statement is optional for `lm_linkage` and `lm_twoqt1`. These programs batch scored realizations in order to provide a Monte Carlo estimate of the standard deviation in estimating the lod score. This statement determines the batch size, and hence the number of batches. By default it is determined such that there are 20 batches.

The following additional statements are specific to tloc specification and likelihood computation for the program `lm_twoqt1`.

```
set traits K1 ... multiple tlocs L1 ...
```

This statement is used by the `lm_twoqt1` program to specify the tlocs *L1*... that contribute to *each* trait *K1*... A statement may be provided for each separate trait. However, the `lm_twoqt1` program expects selection of one trait with either one or two contributing tlocs.

```
set trait K1 for tlocs L1 L2 joint genotype means X11 X12 X13 X21 X22 X23 X31  
X32 X33
```

This statement specifies the 9 genotypic means (3x3 matrix) for tlocs *L1* and *L2* in contributing to trait *K1*. The first index on *X* refers to the *L1* genotype and the second to *L2*.

```
use [exact|MC] summation for trait
```

This statement specifies whether exact or Monte Carlo (MC) will be used by `lm_twoqt1` for computation of the trait contribution to the lod score. Exact summation can be used only on pedigrees with six or fewer founders.

```
use I MC realizations for trait
```

If MC summation is to be used, this statement specifies the number of realizations of tloc inheritance realizations to be used. If MC summation is not to be used, this statement is ignored.

```
use multiplier I MC realizations for null
```

This statement specifies the number of time as many realizations are to be used in estimating the base-line unlinked lod-score. To obtain accurate lod-score estimates it is important this value is accurate, and it may therefore be advisable to use more realizations.

See [\[Concept Index\]](#), page 112 for: location lod scores computational parameters, pseudo-prior.

11.8.8 Location lod scores MCMC parameters and options

All the statements described in see [Section 8.6 \[MCMC parameter statements\]](#), page 49 for specifying the MCMC parameters are used for the location lod scores programs.

Please see that section for details regarding:

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

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

```
check progress I MC iterations
```

As with the Autozgc programs, the number of desired MC iterations must be specified, as there is no default value.

set MC iterations *I*

This statement sets the total number of ‘main’ L- and M-sampler iterations. For `lm_linkage`, the total MCMC run length is the sum of the number of burn-in iterations and main iterations. For `lm_bayes`, the total MCMC run length is the sum of the number of burn-in, pseudo-prior (see below) and main iterations.

Additional statements for `lm_bayes` include the following:

set pseudo-prior iterations *I*

Following burn-in, `lm_bayes` performs iterations to calculate the pseudo-priors. These pseudo-priors are used to encourage the MCMC sampler to visit test positions of low posterior probability. The default number of iterations to compute pseudo-priors is 50% of the number of main iterations specified in the ‘set MC iterations’ statement.

set sequential imputation proposals every *I* iterations

This option applies to `lm_bayes`’s pseudo-prior and main MCMC iterations. It allows the MCMC chain to “restart” every *I*th iteration. Sequential imputation is used to propose potential restart configurations which are accepted/rejected with Metropolis-Hastings probability.

set test position window *I*

This `lm_bayes` statement specifies the window size for the proposed tloc position update in the Metropolis-Hastings algorithm. *I* is the number of hypothesized trait positions on either side of the current position, with equal weight given to the $2*I + 1$ trait positions. The default is window size is 6.

See [\[Concept Index\]](#), page 112 for: location lod scores MCMC parameters and options, MC iterations, burn-in, sequential imputation proposals.

12 Polygenic Modeling of Quantitative Traits by EM Algorithm

See [Concept Index], page 112 for: polygenic model, PolyEM, EM algorithm, quantitative trait.

12.1 Introduction to PolyEM programs

See [References], page 110, for details of the cited papers.

PolyEM is a set of programs to evaluate the likelihood and compute MLEs for polygenic models of quantitative traits by EM algorithms. The original versions of these programs were based on the work described in [TS90] and [TS92].

There are four main programs whose features are summarized below:

- **univar**: This program fits a univariate trait model. It is primarily for test purposes. The likelihood is computed by three methods including classical pedigree polygenic peeling [ES71], which does not extend to looped pedigrees, by direct inversion of the covariance matrix, which does not extend to large pedigrees, and by a general matrix elimination peeling method which is the method used by the other ‘PolyEM’ programs.
- **unibig**: An extension of **univar** to big pedigrees that implements more efficient methods to compute the polygenic likelihood on large looped pedigrees.
- **bivar**: An extension of **unibig** for bivariate traits.
- **multivar**: An extension of **bivar** for multivariate traits.

All programs can work with looped pedigrees. The exception is that looped pedigrees cannot be used for the polygenic peeling algorithm in **univar**. The other programs do not use polygenic peeling to evaluate the likelihood.

Only examples and statement references for **multivar** are given since it has the most complete features. The statements for other programs are similar with some exceptions. For example, any statements with between-trait covariances do not apply to **univar** or **unibig**, since these deal only with a single trait.

See [Concept Index], page 112 for: PolyEM introduction, **univar**, **unibig**, **bivar**, **multivar**.

12.2 Sample multivar parameter file

The example pedigree file ‘**polyem.ped**’ for the PolyEM programs is a 90-member pedigree consisting of two 45-member components. The format is similar to ‘**ped73.ped**’, which was used in most of the previous examples.

The first three entries in each line consist of the individual’s name, father’s name and mother’s name. Integers starting with the fourth column (usually gender) can be fixed effects (gender, age class, etc.) or discrete phenotypes.

For quantitative traits, real numbers follow the names and integers. These real numbers represent trait measurements. Missing values are coded with integer part ‘999’, such as 999.5 in the following example.

Here is part of the pedigree file ‘**polyem.ped**’. This file can be found in the ‘PolyEM’ subdirectory of ‘**MORGAN_V30_Examples**’.

```

input pedigree size 90
input pedigree record names 3 integers 3 reals 2
*****
  1   0   0   1   1   0   0.0246  -1.0125
  2   0   0   2   1   0  -0.5978   1.5963
  3   0   0   1   1   0  -0.8124   0.5662
  4   0   0   2   1   0   0.4334   1.7721
  5   1   2   1   1   0   0.1802  -1.4672
  6   1   2   1   1   0  -1.7557   0.8091
  7   3   4   2   1   0   999.5    999.5
  8   3   4   2   1   0   1.9128   0.9780
  9   0   0   2   1   0   0.9530   2.3473
  ...

```

Below is the example multivar parameter file, 'polyem.par'.

```

input pedigree file 'polyem.ped'

select traits 1 2
set trait 2 effects 1 2

start residual covariance  -0.09
start additive covariance  -0.0017
start residual variance    1.10  0.65
start additive variance    0.037  0.0288

fit residual covariance
  1

fit additive covariance
  1

fit environmental model

output spacing 20 EM iterations
limit EM iterations 200

```

`multivar` can fit a polygenic model with one to five traits, which can be modeled as dependent and/or independent. One '`select traits`' statement must be given. The number of integer values entered as arguments to the statement must be the number of quantitative traits expected by the program being run. For example, for programs `univar` and `unibig`, the '`select traits`' statement must have a single integer argument. When running `bivar`, two integer arguments must be given. For `multivar`, one to five integer arguments must be given in the '`select traits`' statement, for the one to five quantitative traits selected. Unlike other MORGAN programs, the trait numbers correspond to the column number of the reals in the pedigree file. In the example, the statement '`select traits 1 2`' indicates that the first two column of real numbers contain the trait data to be analyzed.

The statement '`set trait 2 effects 1 2`' indicates that the second column of real numbers is to be modeled with two *fixed effects* (also called *covariates*). The integers give the location

of the fixed effects (covariates) *starting with column 4* in the pedigree file. In this example, the fixed effects are to be found in columns 4 and 5. **Important:** a fixed effect location of ‘1’ indicates that the effect value will be found in column 4 (*after* the 3 name columns). The most commonly modeled fixed effect is gender, which, if present, resides in column 4 of a MORGAN pedigree file.

The statement ‘`start trait mean`’ allows the user to specify the starting trait mean for a selected trait. Since no ‘`start trait mean`’ statement is included after either of the ‘`select trait`’ statements, both of the initial means are computed by the PolyEm program. Similarly, one may specify the initial values for each effect with the statement ‘`start trait I effect M X1 X2 ...`’, where ‘I’ is the trait number, ‘M’ is the effect number and ‘Xi’ is the starting value of the *i*th level (*i* = 1, 2, 3, ...). These starting values represent deviations from the global mean. The starting values are normalized so that their weighted sum is zero (weighted by the number of individuals in that level). When using the ‘`start trait I effect M X1 X2 ...`’ it is important to know that if more levels are present in the column of numbers corresponding to the trait ‘I’ in the pedigree file than are specified in the ‘`start trait`’ statement, PolyEm programs will compute the starting value(s) for these additional levels. Since the program will not issue a warning or error message, it is important to always check the output to confirm that the number of levels present in the file was as intended. Since the ‘`start trait I effect M X1 X2 ...`’ statement is not included in this example, the PolyEm program will compute the initial values of the effect.

Initial values for additive and residual variances and covariances are specified in the next four statements. These statements are required. With the variance statements, the number of arguments must be the same as the number of traits selected and must be in order of increasing trait number. With the covariance statements, the number of arguments must be the same as the number of pairs of traits selected. See [Section 12.4.2 \[multivar segregation model parameters\]](#), page 100, for discussion of the ordering of these arguments.

`multivar` can also fit a purely environmental model with no genetic component. The ‘`fit environmental model`’ statement tells `multivar` to fit a purely environmental model, with no genetic variance. This *null hypothesis* model is produced in addition to the genetic/environmental model.

The final two statements specify the number of EM iterations and how often the EM estimates are to be printed out.

Note that one has the opportunity to provide predetermined eigenvalues of the G-matrix of observed individuals. The ‘`set eigenvalues`’ statement is used to specify the eigenvalues, with the number of values equal to the number of observed individuals. If desired, the eigenvalues can be provided through an input file accessed with a ‘`input eigenvalue file`’ statement in the parameter file, or through the command line (see [Section 12.4.3 \[multivar computational parameters\]](#), page 101).

See [\[Concept Index\]](#), page 112 for: `multivar` sample parameter file, missing quantitative trait data.

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’ statement in the parameter file.

Under the subdirectory ‘PolyEM/’, run the example by typing:

```
./multivar polyem.par
```

Toward the end of the *multivar* output, are the parameter estimates and the log-likelihood from the last iteration of the EM algorithm. If you chose to fit a null (purely environmental) model, using the ‘fit environmental model’ statement, those parameter estimates and log-likelihood are also given. A likelihood ratio test can then be performed, with test statistic equal to the absolute value of 2 times the difference between the log-likelihoods of the two models. A conservative test is provided by comparing the test statistic to a chi-squared distribution, with the degrees of freedom being the difference in the numbers of estimated parameters between these two models. Note that in these ‘Polyem’ programs, model log-likelihoods are in base *e* rather than the usual lod score base-10 convention.

```
iteration #201:

additive variance estimates (traits 1, 2)
    0.816    0.037
covariances
    0.138

residual variance estimates (traits 1, 2)
    0.223    0.610
covariances
    -0.239

trait 1
  overall mean      -0.063
trait 2
  overall mean       1.780
  fixed effect  1   -0.717    0.546
  fixed effect  2   -1.008   -0.552    1.167

current log-likelihood = -183.098

estimates of environmental model

residual variance estimates (traits 1, 2)
    1.136    0.642
covariances
    -0.102

trait 1
  overall mean       0.062
trait 2
  overall mean       1.801
```

```

fixed effect 1  -0.773    0.589
fixed effect 2  -1.010   -0.553    1.170

```

```
environmental model log-likelihood = -197.799
```

In this example, we see that the fitted genetic model has a very significantly larger log-likelihood: $2(-183.098 + 197.799) = 29.4$, with 3 extra genetic variance parameters fitted. The estimates of the fixed effects are little altered by fitting the genetic model but for trait 1 the additive genetic variance is large relative to the residual variance, indicating a strong genetic component. Note that, for each trait, the sum of the additive and residual variances from the genetic model is close to the residual variance in the environmental model.

See [Concept Index], page 112 for: running `multivar` example, `multivar` sample output.

12.4 multivar statements

See [Concept Index], page 112 for: PolyEM statements, `multivar` statements.

12.4.1 multivar computing requests

```
select traits I1...
```

One ‘`select trait`’ statement is required, and must list the traits to be modeled. Up to five traits are allowed for `multivar`. The trait number, I , corresponds to the column of real numbers in the pedigree file, with the first column of real numbers being trait 1, the second column trait 2 and so on.

```
set trait I effects M1...
```

$M1\dots$ are the fixed effects to be modeled for a specified trait I . They are the integer columns in which they appear in the pedigree file. That is the columns after the three names, so that fixed effect ‘1’ is in the 4th column (usually *gender*), effect ‘2’ is in column 5, and so on.

See [Concept Index], page 112 for: `multivar` computing requests.

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: X_{12} , X_{13} , ..., X_{1n} , X_{23} , ..., X_{2n} , ..., where X_{ij} is the covariance for the i th and j th selected traits from the pedigree file.

`start residual (covariances | correlations) X12 ...`

See the ‘`start additive...`’ statements.

See [Concept Index], page 112 for: multivar segregation model parameters.

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.

See [Concept Index], page 112 for: multivar computational parameters, observed individuals.

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 addition to the genetic/environmental model.

See [\[Concept Index\]](#), page 112 for: multivar computational options.

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 *I*th iteration. The default number is defined in the program header file.

`check gmatrix`

This statement requests a print out of the G matrix for observed individuals and quit without doing the likelihood computation.

`check ginverse`

This statement requests a print out of the G inverse matrix and the program quits unless ‘`check eigenvalues`’ has also been specified.

`check eigenvalues`

This statement requests the program to print out of the eigenvalues, whether computed or input, and then to quit. These eigenvalues can then be used as input in subsequent runs.

`check eigenvalue computation`

This statements causes some comments to be printed by the function that computes the eigenvalues.

`check trace`

This statements requests the trace of the G-inverse matrix to be printed.

See [\[Concept Index\]](#), page 112 for: `multivar` output options.

13 Estimating Genetic Maps from Marker Data

See [\[Concept Index\]](#), page 112 for: genetic map estimation.

13.1 Introduction to `lm_map`

See [\[References\]](#), page 110, for details of the cited papers.

The program `lm_map` finds the maximum likelihood estimate (MLE) of the marker map, estimates the (statistical not Monte Carlo) variance of the MLE, and tests hypotheses about the true map. All inference is based on the analysis of multilocus marker data obtained from some (possibly all) members of a set of independent families (pedigree components).

To find the MLE, `lm_map` uses either Monte Carlo expectation-maximization (MCEM) or a hybrid of MCEM and stochastic approximation (SA). In either case, the user must supply an initial map estimate, and an initial Monte Carlo (MC) sample size for the MCEM algorithm. The MCEM sample size is automatically increased with each successive step of the algorithm, and only a small number of MCEM steps are needed to estimate the MLE. If the hybrid option is chosen, `lm_map` uses the MCEM estimate to seed the SA algorithm. Then, a relatively large number of SA steps are used to estimate the MLE with greater precision.

Once the MLE is obtained, a long Markov chain is used to estimate the variance of the MLE. Finally, a slight adaptation of the MC likelihood ratio formula is used to estimate the likelihood ratio test (LRT) statistics for testing the simple and/or composite null hypotheses. For more details, see [ST06].

See [\[Concept Index\]](#), page 112 for: `lm_map` introduction.

13.2 Sample `lm_map` parameter file

The two sample parameter files for `lm_map` can be found in the directory 'MORGAN_V30_Examples/Map'. The two files are 'map_G.par' and 'map_P.par', along with the corresponding marker data files 'map_G.markers' and 'map_P.markers'. Thus there are two examples, one for genotypic markers (G) and one for phenotypic markers (P). 'G' denotes that marker genotypes are observed without error. 'P' denotes the possibility of error, so that the observed marker phenotype is not the same as the underlying true marker genotype. This example uses the pedigree file 'map.ped', but different marker data files depending on the choice of 'P' or 'G'.

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

```
input pedigree file './map.ped'
input marker data file './map_G.markers' # or './map_P.markers' for 'map_P.par'

select all markers
set marker 1 2 3 allele freqs .2 .2 .2 .2 .2
set marker names DS123 DS456 DS789

map gender F marker recomb fract .18 .18 # true F map (cM): 20 20
map gender M marker recomb fract .08 .08 # true M map (cM): 10 10
```

```

limit recomb fracts .001

use sequential imputation for setup
use 100 sequential imputation realizations for setup
set burn-in iterations 100
sample by scan
set L-sampler probability .8
set MC iterations 50 # The initial number of MCMC scans per step
limit EM iterations 10 # The total number of MCEM steps

```

As seen in previous examples, the ‘select all markers’ statement instructs the program to use all markers on the chromosome for computation. The alternative is to use only selected markers for computation, which can be achieved by using the ‘select markers’ statement (see [Section 9.8.1 \[Autozyg computing requests\]](#), page 61). The ‘set marker 1 2 3 allele freqs .2 .2 .2 .2’ statement specifies the marker allele frequencies for markers 1, 2, and 3. This statement, as constructed, requires markers 1, 2, and 3 to each have five alleles with frequencies of 0.2 for each allele. If the number of alleles per marker varies from marker to marker, or if the allele frequencies vary from marker to marker, a separate ‘set marker freqs’ statement is needed for each marker (see [Section 6.5.3 \[markerdrop population model parameters\]](#), page 37). The ‘set marker names’ statement overrides the default behavior, which labels markers consecutively: marker-1, marker-2, etc.

The two ‘map gender marker recomb fract’ statements specify the marker map in terms of recombination fractions. This is the initial starting estimate of the map.

The ‘limit recomb fract_s 0.001’ statement is optional and places lower and upper bounds on the estimated recombination fractions of the map. For markers that are separated by little or no recombination, the MCEM algorithm may yield estimated recombination fractions of zero which could lead to a severe bias in the results. As a safeguard against such events, this statement places a lower bound 0.001 and an upper bound 0.5 - 0.001 on the estimated recombination fractions of the map.

The statement ‘use sequential imputation for setup’ instructs `lm_map` to initialize the set of maternal and paternal meiosis indicators for all members of the pedigree who are not founders; this is done prior to the Monte Carlo simulation. The default behavior is specified in this statement, with the alternative being to ‘use locus-by-locus sampling for setup’. The statement ‘use 100 sequential imputation realizations for setup’ is optional and modifies the default behavior for setup by sequential imputation (which is 10% of the MC iterations). The next three lines in the parameter files contain statements introduced in the `Autozyg` examples of this tutorial. For explanation of ‘set burn-in iterations’, ‘sample by scan’, and ‘set L-sampler probability’ see [Section 9.8.8 \[Autozyg MCMC parameters and options\]](#), page 65. The statement ‘set MC iterations 50’ indicates how many MC iterations are to be performed at each EM iteration. The statement ‘limit EM iterations’ was introduced in the `multivar` example and puts an upper bound on the number of MCEM iterations.

Now we’ll take a look at the remaining statements in ‘map_G.par’:

```

output maps gender averaged specific
set map estimation model with no mistyping
set EM convergence .01

```

```

use MCEM and SA for maximization
set SA curvature iterations 10
set SA ascent iterations 10
set SA gradient iterations 10
set SA convergence .001

```

The ‘output maps gender averaged specific’ statement specifies the type of map to be estimated by `lm_map`. In this example, the default behavior is specified, which instructs `lm_map` to automatically compute the likelihood ratio test statistic for testing the null hypothesis of a sex-averaged map. The statement ‘set map estimation model with no mistyping’ instructs `lm_map` to assume that the genotypes are observed without error. The ‘set EM convergence’ statement instructs `lm_map` to stop the MCEM algorithm if all recombination fraction updates are within 0.01 of their previous values.

The statement ‘use MCEM and SA for maximization’ in ‘map_G.par’ instructs `lm_map` to attempt to refine its MCEM-based estimate of the MLE by performing additional SA steps. The alternative is to ‘use MCEM only for maximization’, with no further refining. There are several statements that allow additional control of the SA algorithm. First, an estimate of the curvature of the likelihood is needed to initiate the SA algorithm. The statement ‘set SA curvature iterations 10’ instructs `lm_map` to use at least 10 MCMC realizations to estimate the curvature of the likelihood. Also, `lm_map` will not initiate the SA algorithm with a step that decreases likelihood. So, when the SA algorithm is used for refining the likelihood estimate, the statement ‘set SA ascent iterations 10’ instructs `lm_map` to use at least 10 MCMC realizations to determine whether a proposed first step increases the likelihood. The SA algorithm also requires an estimate of the gradient of the likelihood at each SA step. The statement ‘set SA gradient iterations 10’ instructs `lm_map` to use at least 10 MCMC realizations to estimate the gradient of the likelihood. Finally, the map estimate obtained from the final step of the MCEM algorithm is used to seed the SA algorithm. The ‘set SA convergence 0.001’ statement instructs `lm_map` to terminate the SA algorithm when the absolute change in successive map estimates is less than 0.001 for each recombination fraction in the map.

The file ‘map_P.par’ shows some different Monte Carlo and estimation options. Here are the remaining statements in that file:

```

output maps gender averaged
set map estimation model with mistyping
set genotyping error rate .02
use MCEM only for maximization

```

In this parameter file, a gender averaged map is specified by using the ‘output maps gender averaged’ statement. Unlike in the previous parameter file, ‘map_P.par’ does not assume the genotypes are recorded without error; this is indicated by the statement ‘set map estimation model with mistyping’. When ‘with mistyping’ is chosen, one has the option of specifying an estimate of the error rate with the statement ‘set genotyping error rate E ’. In this example, the error rate is set at 0.02. Finally, the statement ‘use MCEM only for maximization’ instructs `lm_map` not to use the SA algorithm to further refine the MCEM-based estimate of the MLE. Since the SA algorithm will not be used, none of the ‘SA’ statements are used in ‘map_P.par’.

See [\[Concept Index\]](#), page 112 for: sample parameter file for `lm_map`.

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

See [Concept Index], page 112 for: running `lm_map` with genotypic data, `lm_map` sample output for genotypic data.

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.

See [Concept Index], page 112 for: running `lm_map` with phenotypic data, `lm_map` sample output for phenotypic data.

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 by little or no recombination, the MCEM algorithm may yield estimated recombination fractions of zero which could lead to a severe bias in the results. As a safeguard against such events, this statement places a lower bound L and an upper bound $0.5 - L$ on the estimated recombination fractions of the map.

`output maps gender [averaged] [specific]`

This statement specifies the type of map to be estimated. The default behavior is to select both options, which instructs `lm_map` to automatically compute the likelihood ratio test statistic for testing the null hypothesis of a sex-averaged map.

`use MCEM and SA for maximization`

If the statement ‘use MCEM only for maximization’ is replaced by this statement, `lm_map` will attempt to refine its MCEM based estimate of the MLE by performing additional SA steps.

`set EM convergence X`

The MCEM algorithm is used to find a suitable starting value for the SA algorithm. The MCEM algorithm terminates when the percent change in successive

parameter estimates is less than X . The default value of X is 0.2: smaller values may substantially increase the total CPU time.

set genotyping error rate E

When the statement ‘**set map estimation with mistyping**’ is used, the genotype observations are assumed to have an associated error rate. This statement allows for the specification of the ‘**mistyping**’ rate.

set SA curvature iterations I

An estimate of the curvature of the likelihood is needed to initiate the SA algorithm. This statement tells `lm_map` to use at least I MCMC realizations to estimate the curvature of the likelihood. The curvature is only estimated once.

set SA ascent iterations I

`lm_map` will not initiate the SA algorithm with a step that decreases the likelihood. This statement tells `lm_map` to use at least I MCMC realizations to determine whether a proposed first step increases the likelihood.

set SA gradient iterations I

If SA is initiated, this tells `lm_map` to use at least I MCMC realizations to estimate the gradient of the likelihood. An estimate of the gradient is needed for each SA step.

set SA convergence R

The SA algorithm is terminated, if all recombination fraction updates are within R of their previous values. In addition, the maximum possible runtime for the SA algorithm is proportional to the total runtime of the MCEM algorithm.

set map estimation (with | with no) mistyping

This statement can be used to specify whether or not errors were made during the observation of genotype. If ‘**with no**’ is selected, the genotypes are assumed to have been observed without error. If ‘**with**’ is selected, the genotype observations are assumed to have some error associated with them, which can be specified using the ‘**set genotyping error rate**’ statement.

set LRT statistics iterations I

This statement tells `lm_map` to use at least I MCMC realizations to estimate the LRT statistics. If only one option is used in ‘**output maps gender . . .**’, then the estimated LRT statistic compares the MLE to the initial map. Otherwise, two LRT statistics are estimated. The first compares the MLE of the sex-averaged map to the initial sex-averaged map, while the second compares the MLE of the sex-specific map to the MLE of sex-averaged map.

compute estimates I times

This statement tells `lm_map` to conduct its entire analysis I times, and to report the map with the highest likelihood. While this statement offers some protection against convergence to local modes, the default value is 1.

See [\[Concept Index\]](#), page 112 for: `lm_map` statements.

References

- [Bas08] Basu, S., Di, Y., and Thompson, E. A. (2008), “Exact trait-model-free tests for linkage detection in pedigrees,” *Annals of Human Genetics* **72**, 672–682.
- [Bas10] Basu, S., Stephens, M., Pankow J. S., and Thompson, E. A. (2010), “A likelihood-based trait-model-free approach for linkage detection of binary trait,” *Biometrics* **66**, 205–213.
- [Bau72] Baum, L E. (1972), “An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes,” in O. Shisha, ed., ‘Inequalities-III; Proceedings of the Third Symposium on Inequalities. University of California Los Angeles, 1969’, Academic Press, New York, pp. 1–8.
- [Don83] Donnelly, K. P. (1983), “The probability that related individuals share some section of genome identical by descent,” *Theoretical Population Biology*, **23**, 34–63.
- [DT09] Di, Y. and Thompson, E. A. (2009), “Conditional tests for localizing trait genes,” *Human Heredity* **68**, 139–150.
- [ES71] Elston, R. C. and Stewart, J. (1971), “A general model for the analysis of pedigree data,” *Human Heredity* **21**, 523–542.
- [GT03] George, A. W. and Thompson, E. A. (2003) “Discovering disease genes: Multipoint linkage analysis via a new Markov Chain Monte Carlo approach,” *Statistical Science* **18**, 515–531.
- [GWT05] George, A. W., Wijsman, E. M. and Thompson, E. A. (2005) MCMC Multilocus Lod Scores: Application of a New Approach. *Human Heredity* **59**, 98–108.
- [GYO96] Goddard, K. A., Yu, C. E., Oshima, J., Miki, T., Nakura, J., Piussan, C., Martin G. M. et. al. (1996), “Toward localization of the Werner syndrome gene by linkage disequilibrium and ancestral haplotyping: lessons learned from analysis of 35 chromosome 8p11.1-21.1 markers,” *American Journal of Human Genetics*, **58** 1286–1302.
- [Hal19] Haldane, J. B. S. (1919), “The combination of linkage values and the calculation of distances between the loci of linked factors,” *Journal of Genetics*, **8** 229–309. ‘
- [He97] Heath, S.C. (1997). Markov chain Monte Carlo segregation and linkage analysis for oligogenic models. *American Journal of Human Genetics*, 61: 748-760.
- [LG87] Lander, E. S. and Green, P. (1987), “Construction of multilocus genetic linkage maps in humans,” *Proceedings of the National Academy of Sciences (USA)* **84**, 2363–2367.
- [Sie05] Sieh, W., Basu, S., Fu, A. Q., Rothstein, J. H., Scheet, P. A., Stewart, W. C. L., Sung, Y. J., Thompson, E. A., and Wijsman, E. M. (2005), “Comparison of marker types and map assumptions using Markov chain Monte Carlo-based linkage analysis of COGA data,” *BMC Genetics* **6** (Suppl 1), S11.
- [ST06] Stewart, W. C. L. and Thompson, E. A. (2006), “Improving estimates of genetic maps: A maximum likelihood approach,” *Biometrics* **62**, 728–734.
- [STW07] Sung, Y. J., Thompson E. A., and Wijsman E. M. (2007), “MCMC-based linkage analysis for complex traits on general pedigrees: multipoint analysis with a two-locus model and a polygenic component,” *Genetic Epidemiology* **31**, 103–114.

- [SW07] Sung, Y. J. and Wijsman, E. M. (2007), “Accounting for epistasis in linkage analysis of general pedigrees,” *Human Heredity* **63**, 114–152.
- [TG07] Thompson, E. A., and Geyer, C. J. (2007), “Fuzzy p-values in latent variable problems,” *Biometrika* **90**, 49–60.
- [Tho76] Thompson, E. A. (1976), “Peeling programs for zero-loop pedigrees,” Technical Report No. 5, Department of Medical Biophysics and Computing, University of Utah.
- [Tho77] Thompson, E. A. (1977), “Peeling programs for pedigrees of arbitrary complexity,” Technical Report No. 6, Department of Medical Biophysics and Computing, University of Utah.
- [Tho00] Thompson, E. A. (2000), “Statistical Inference from Genetic Data on Pedigrees,” in ‘NSF-CBMS Regional Conference Series in Probability and Statistics,’ Vol. 6, Institute of Mathematical Sciences, Beachwood, Ohio.
- [Tho08] Thompson, E. A. (2008), “Uncertainty in inheritance: assessing linkage evidence,” in ‘Joint Statistical Meetings Proceedings,’ Salt Lake City 2007, pp. 3751–3758.
- [Tho11] Thompson, E. A. (2011), “The structure of genetic linkage data: from LIPED to 1M SNPs,” *Human Heredity* **71** 86–96.
- [TH99] Thompson, E. A. and Heath, S. C. (1999). Estimation of conditional multilocus gene identity among relatives. *Statistics in Molecular Biology and Genetics*, Seillier-Moseiwitch F., (ed). IMS Lecture Note - Monograph Series, **33**: 95-113.
- [TS90] Thompson, E. A. and Shaw, R. G. (1990). Pedigree analysis for quantitative traits: variance components without matrix inversion. *Biometrics*, **46**: 399-413.
- [TS92] Thompson, E. A. and Shaw, R. G. (1992). Estimating polygenic models for multivariate data on large pedigrees. *Genetics*, **131**: 971-978.
- [TT08] Tong, L. and Thompson, E. A., “Multilocus lod scores in large pedigrees: Combination of exact and approximate calculations,” *Human Heredity* **65**, 142–153.

Concept Index

A

<i>a priori ibd</i> probabilities	39
additive variance	22, 24, 27
affected individuals	65
allele frequencies	27
allow statements	10
Autozyg computational parameters	64
Autozyg computing requests	61
Autozyg file identification statements	62
Autozyg mapping model parameters	63
Autozyg MCMC parameters and options	65
Autozyg output file description	63
Autozyg pedigree file description	63
Autozyg population model parameters	64
Autozyg programs	51
Autozyg statements	61
autozygosity	51

B

bivar	96
block Gibbs sampler	46
burn-in	48, 49, 54

C

civil introduction	67
civil sample output	73
civil statements	74
command line options	7
command syntax	7
component	15, 16, 17, 20
conditional <i>ibd</i> probabilities	51
consistency (Mendelian) of marker data	61
continuous trait specification for lod score calculation	80

D

debug control	10
descent graph	51
discrete trait	37, 65
discrete trait specification for lod score calculation	79
display options	10

E

EM algorithm	96
exact computation	58
exact computations	45
extra file	9

F

file names	8
file separator	11
file type codes	7
founder gene labels	22, 24, 28, 35, 44
founder genome labels	22, 44, 58, 63
fuzzy p-value	60

G

gender	12, 17
gender-specific maps	26, 53, 78
genedrop computational parameters	28
genedrop computing requests	26
genedrop introduction	22
genedrop mapping model parameters	26
genedrop output pedigree options	28
genedrop output seed file	29
genedrop population model parameters	27
genedrop sample output	25
genedrop sample parameter file	23
genedrop statements	26
genetic map estimation	104
genetic model	44
genotypic trait	53, 65
genotypic trait specification for lod score calculation	79
gl_auto introduction	51
gl_auto sample output	59
gl_auto sample parameter file	57
gl_auto statements	61
GLUT runtime display	10

H

Haldane map function	26, 36, 37, 40, 92
HMM computations	45
how to get the examples	2
how to get the tutorial	2

I

<i>ibd</i>	18, 39, 44, 51
<i>ibd</i> graph	51, 58
<i>ibd</i> measures	74
<i>ibd</i> pattern	39, 41, 56
<i>ibd</i> -based tests	67
ibddrop introduction	39
ibddrop sample output	40
ibddrop sample parameter file	39
ibddrop statements	42
identity by descent	39, 51
inbreeding coefficient	18, 21
incomplete penetrance	37, 65

incomplete penetrances 30
 inheritance indicators 28, 30, 39, 43, 44, 49, 51
 inheritance vector 44

K

kin introduction 18
 kin sample output 19
 kin sample parameter file 18
 kin statements 20
 kinship coefficient 18, 21
 Kosambi map function 26, 36, 37, 92

L

L-sampler 46, 49, 54, 79
 L-sampler probability 47
 latent p-values 51, 70
 likelihood-ratio measures 74
 limit overriding 10
 limit statements 10
 LM-sampler 46, 51
 lm_auto introduction 51
 lm_auto sample output 55
 lm_auto sample parameter file 52
 lm_auto statements 61
 lm_bayes introduction 76
 lm_bayes sample output 82
 lm_bayes statements 91
 lm_ibdttests introduction 67
 lm_ibdttests sample output 69
 lm_ibdttests statements 74
 lm_linkage introduction 76
 lm_linkage sample output 80
 lm_linkage statements 91
 lm_lods statements 91
 lm_map introduction 104
 lm_map sample output for genotypic data 107
 lm_map sample output for phenotypic data 107
 lm_map statements 108
 lm_pval introduction 51
 lm_pval sample output 60
 lm_pval sample parameter file 60
 lm_pval statements 61
 lm_twoqt1 introduction 76
 lm_twoqt1 sample output 83
 location lod scores computational parameters .. 93
 location lod scores computing requests 91
 location lod scores estimates 76
 location lod scores file identification statements
 91
 location lod scores mapping model parameters .. 92
 location lod scores MCMC parameters and options
 94
 location lod scores output file description 92
 location lod scores pedigree file description 92
 location lod scores population model parameters
 93

location lod scores statements 91
 locus window 64
 locus-by-locus sampling 49
 locus-by-locus setup 47

M

M-sampler 46, 49, 54, 79
 major gene model 22
 map test tlocs jointly at markers 83
 marker allele frequencies 37
 marker data 64
 marker data file 9, 53
 marker names 37
 marker seeds 38
 marker simulation 36
 marker simulation using meiosis indicators 36
 marker simulation using trait 36
 markerdrop computational parameters 37
 markerdrop computing requests 36
 markerdrop input file options 38
 markerdrop introduction 30
 markerdrop mapping model parameters 36
 markerdrop output 34
 markerdrop parameter file – conditional on
 inheritance pattern 33
 markerdrop parameter file – conditional on trait
 30
 markerdrop population model parameters 37
 markerdrop statements 35
 Markov chain Monte Carlo 44, 51, 76
 MC iterations 48, 49, 95
 MCMC introduction 44
 MCMC options 47
 MCMC parameter statements 49
 meiosis indicators 28, 30, 39, 43, 44, 49, 51, 76,
 79
 missing marker data 64
 missing quantitative trait data 96
 mixed model 22
 MORGAN 1
 MORGAN files 7
 MORGAN Gold standards 4
 MORGAN program libraries 4
 MORGAN subroutine libraries 4
 multiple meiosis sampler 46
 multiple meiosis sampler 46, 51, 58, 76, 79
 multivar 96
 multivar computational options 102
 multivar computational parameters 101
 multivar computing requests 100
 multivar output options 102
 multivar sample output 98
 multivar sample parameter file 96
 multivar segregation model parameters 100
 multivar statements 100

N

names of individuals 12
 non-genetic model 22

O

observed individuals 12, 101
 output control 10
 output warnings (W) 10
 overview of MORGAN 1
 overwrite file options 9, 16

P

parameter file 8
 parameter statements 8
 parameter statements in pedigree files 11
pedcheck examples 16
pedcheck introduction 15
pedcheck sample parameter file 15
pedcheck statements 17
 pedigree component 16, 17, 20
 pedigree file 9, 11
 pedigree file descriptions 12
 pedigree options 17
 pedigree order 17
 pedigree peeling 46
 pedigree record format 12, 38
 pedigree size 12
 pedigree trait integer data order 13
 pedigree trait real data order 13
 pedigree validity 15
 penetrance 37, 44
 phenotypic trait specification for lod score
 calculation 79
 PolyEM 96
 PolyEM introduction 96
 PolyEM statements 100
 polygenic model 22, 96
 printlevel control 10
 proband gametes 40, 43, 54, 56, 64
 pseudo-prior 93
 pseudo-prior iterations 48

Q

quantitative trait 22, 65, 92, 96
 quantitative trait specification for lod score
 calculation 80

R

randomized p-values 70
 Rao-Blackwellized estimates 82, 91
 README documentation files 4
 recombination 21
 redirect output 10

rescue file 62
 residual variance 24, 27
 running **genedrop** examples 25
 running **gl_auto** example 59
 running **ibddrop** example 40
 running **kin** example 19
 running **lm_auto** example 55
 running **lm_bayes** examples 82
 running **lm_linkage** examples 80
 running **lm_map** with genotypic data 107
 running **lm_map** with phenotypic data 107
 running **lm_pval** example 60
 running **lm_twoqtl** examples 83
 running **markderdrop** examples 34
 running **multivar** example 98

S

sample by scan 47, 49
 sample by step 47, 49
 sample parameter file for **civil** 70
 sample parameter file for **lm_bayes** 77
 sample parameter file for **lm_ibdttests** 68
 sample parameter file for **lm_linkage** 77
 sample parameter file for **lm_map** 104
 score file 62
 scoreset 64
 seed file 9, 23, 40
 seeds for data simulation 22, 25, 28, 29
 seeds for sampler 40, 43
 sequential imputation 47, 49
 sequential imputation proposals 95
 set printlevel statement 30
 simulating marker data 22, 28
 simulating trait data 22, 28
 specifying inheritance 44
 structure of the MORGAN package 4

T

trait allele frequencies 37
 trait data 37, 53, 65
 trait model 30
 trait seeds 38
 trait test positions 92
 Tutorial for lod score examples with ‘ped73.ped’
 3

U

unibig 96
univar 96
 unobserved individuals 12, 22, 28

V

vc-score and w-score 67

W

white space 7
whitespace 11

window patterns 53, 64

Statement Index

A

allow component size.....	10
allow observed individuals.....	10
allow pedigree size.....	10, 12
assign gender.....	17
assume all observed.....	29

C

check eigenvalue computation.....	102
check eigenvalues.....	102
check ginverse.....	102
check gmatrix.....	102
check marker consistency [only].....	62
check progress MC iterations.....	49
check trace.....	103
compute [component] inbreeding coefficient	21
compute [component] kinship coefficient... ..	21
compute [component] two-locus inbreeding coefficient.....	21
compute eigenvalues.....	102
compute estimates I times.....	109
compute ibd statistics.....	74
compute likelihood-ratio statistics.....	74
compute scores every iterations.....	49

F

fit additive correlations.....	101
fit additive covariances.....	101
fit environmental model.....	102
fit residual correlations.....	101
fit residual covariances.....	101

I

ignore gender.....	17
input eigenvalue file.....	101
input extra file.....	7, 9, 75
input marker data file.....	7, 9
input pedigree file.....	7, 9
input pedigree record (father mother father mother).....	12
input pedigree record gender (present absent).....	12
input pedigree record inheritance integer pairs.....	38
input pedigree record names 3 [integers] [reals].....	12
input pedigree record observed (absent present).....	12, 28
input pedigree record traits integers ..	13, 38
input pedigree record traits reals.....	13, 92

input pedigree size.....	12
input rescue file.....	62
input seed file.....	7, 9

L

limit breeding iterations.....	101
limit EM iterations.....	101
limit recombination fractions L.....	108

M

map [chromosome] test tloc at markers.....	93
map [chromosome] test tlocs jointly at markers	93
map [chromosome] [gender] marker [Kosambi] distances.....	26
map [chromosome] [gender] marker [Kosambi] positions.....	26
map [chromosome] [gender] marker recombination fractions.....	26
map [chromosome] [gender] test tloc all interval proportions.....	92
map [chromosome] [gender] test tloc beginning [Kosambi] distance.....	92
map [chromosome] [gender] test tloc beginning recombination fractions.....	92
map [chromosome] [gender] test tloc ending [Kosambi] distance.....	92
map [chromosome] [gender] test tloc ending recombination fractions.....	92
map [chromosome] [gender] test tloc external distance.....	92
map [chromosome] [gender] test tloc external Kosambi distance.....	92
map [chromosome] [gender] test tloc external recombination fractions.....	92
map [chromosome] [gender] test tloc intervals proportions.....	92
map [chromosome] [gender] tlocs marker [Kosambi] distances.....	26
map [chromosome] [gender] tlocs marker [Kosambi] positions.....	26
map [chromosome] [gender] tlocs marker recombination fractions.....	26
map [gender] marker [Kosambi] distance.....	36
map [gender] marker [Kosambi] position.....	36
map [gender] marker recombination fraction	36
map [gender] tlocs marker [Kosambi] distance	37
map [gender] tlocs marker recombination fraction.....	37
map test tlocs no default external proportions	92

map test tlocs no default interval proportions
 92
 map tlocs unlinked 26, 43, 62

O

output [overwrite] extra file 9
 output [overwrite] pedigree file 9
 output [overwrite] score file 9, 62
 output [overwrite] seed file 9
 output final adjusted phenotypes 102
 output founder genome labels 63
 output maps gender [averaged] [specific]
 108
 output marker seeds only 29
 output meiosis indicators 63
 output pedigree chronological 17
 output pedigree file 7
 output pedigree record father mother 17
 output pedigree record founder gene labels
 28
 output pedigree record mother father 17
 output pedigree record trait latent variables
 28
 output pedigree record unobserved variables
 28
 output permutation seeds only 74
 output Rao-Blackwellized estimates file ... 91
 output rescue data iterations 63
 output rescue file 62
 output sampler seeds only 74
 output scores every scored MC iterations ... 63
 output seed file 7
 output spacing EM iterations 102
 output statistics correlations 102
 output statistics covariances 102
 output trait seeds only 29

S

sample by scan 49
 sample by step 49
 select [chromosome] all markers 61
 select [chromosome] markers 61
 select inheritance 36
 select trait 36, 61, 65, 100
 set [chromosome] locus window 43, 64
 set [chromosome] marker allele frequencies
 27, 37
 set [chromosome] marker names 37, 64
 set [component] [scoreset] window patterns
 64
 set [component] proband gametes 43, 64
 set [component] scoreset proband gametes ... 64
 set batches MC variance estimation 93
 set breeding convergence 101
 set burn-in iterations 49
 set debug level 11

set debug *libname* 11
 set debug main 11
 set eigenvalues 101
 set EM convergence 108
 set genotyping error rate 109
 set global MCMC 50
 set ibd measures 74
 set ibd permutations 74
 set ibd tests 74
 set inheritance tlocs 36
 set L-sampler probability 49
 set likelihood-ratio lambda-p model
 gridpoints 74
 set likelihood-ratio measures 74
 set likelihood-ratio tests 74
 set limit for exact computation 50
 set LRT statistics iterations 109
 set map estimation with [no] mistyping ... 109
 set marker data 64
 set marker seeds 28
 set MC iterations 43, 49, 95
 set MCMC markers only 66
 set normalized allele frequencies 27
 set permutation seeds 75
 set printlevel 11
 set pseudo-prior iterations 95
 set pseudo-priors 93
 set recombination frequencies 21
 set SA ascent iterations 109
 set SA convergence 109
 set SA curvature iterations 109
 set SA gradient iterations 109
 set sampler seeds 43
 set sequential imputation proposals every
 iterations 95
 set test position window 95
 set tlocs allele frequencies 27, 37
 set trait data 65
 set trait data discrete 37
 set trait effects 100
 set trait for tlocs joint genotype means ... 94
 set trait seeds 28
 set traits additive variance 27
 set traits for tlocs genotype means 27
 set traits for tlocs incomplete penetrances
 37, 65
 set traits multiple tlocs 94
 set traits residual variance 27
 set traits tlocs 26, 36, 62
 simulate [chromosome] markers 26
 simulate markers 36, 43
 simulate tloc 43
 simulate traits 26
 start additive correlations 101
 start additive covariances 101
 start additive variances 100
 start residual correlations 101
 start residual covariances 101

start residual variances 100
start trait effect 100
start trait mean 100

U

use exact summation for trait 94
use full EM 102
use locus-by-locus sampling for setup 49
use MC realizations for trait 94
use MC summation for trait 94
use MCEM and SA for maximization 108
use multiplier MC realizations for null 94
use partitioned EM 102
use sequential imputation for setup 49
use sequential imputation realizations for
setup 49