Monte Carlo Programs for Pedigree Analysis; 1990-93

E. A. Thompson,

Department of Statistics, GN-22
University of Washington,
Seattle, Washington 98195
U. S. A.

March, 1994.

Abstract and Acknowledgement

This documentation describes programs for Markov chain Monte Carlo likelihood analyses of genetic traits developed and implemented in the Department of Statistics, University of Washington, between 1990 and 1993. The details of the methodology can be found in the cited journal papers. This documentation describes the progams in their current form; many are under continuing development. The development of the methodology underlying these programs, and the program implementation and documentation, are supported by NIH grant GM-46255.

The development of these programs owes much to former students. Nuala Sheehan first implemented the pedigree neighbourhood structures and Gibbs sampling for simple Mendelian traits. Charles J. Geyer modified and extended this code, and implemented his Metropolis coupled Markov chain Monte Carlo methods. Sun Wei Guo extended the code to the analysis of quantitative traits, and implemented Monte Carlo EM algorithms and Monte Carlo estimates of likelihood ratios. Shili Lin has developed and implemented methods that allow sampling of genotypes at multiallelic loci and to several loci, extending the applicability of the methods in the area of linkage analysis.

Charlie Geyer has continued testing, porting, and development of the programs, in the context of his more general research on Markov chain Monte Carlo and Monte Carlo likelihood, and in so doing has found bugs that could cause errors on other machines or operating systems. He has continued to answer questions relating to the implementation of his methods in the genetic analysis context. In both these ways, he has contributed greatly to the reliability, portability, and capability of the current code. Adam Olshen and Ellen Wijsman have been involved in running and testing the methods and programs; their input has much improved the usability of some of the programs.

Special acknowledgement for this documentation goes to Hannah Payne. She developed the overall structure of the documentation, produced the first drafts of many sections, asked all the questions as a computer-literate non-expert, and documented the answers.

Table of Contents

1.	Introduction	3
	Figure 1; The directory structure	5
2.	Getting Started	6
3.	Makefiles	8
4.	The Main Programs – An Overview	11
	PolyEM, MIXED, LINK_LIN, Autozyg, Clip, Gibbs and McGibbs.	
5.	Header files and data structures	13
6.	The Subroutine Libraries	15
	Nghds, Quant, Mendel_Gibbs, McLib and Rans.	
7.	PolyEM programs – *univer* and *bivar*	20
8.	MIXED programs – *mixd, linkreg* and *mixt_mixd*	27
	Figure 2; The pattern of EM iterations.	
9.	LINK_LIN programs – development of *linkreg*	34
	(information supplied by Shili Lin)	
10.	The Clip Program	37
11.	The McGibbs programs and McLib library	37
	(information supplied by Charles J. Geyer)	
12.	The Data	40
13.	The programs on a VMS machine	46
	(information supplied by Ellen Wijsman)	
	References	48

1 Introduction

1.1 The scope of the programs

This documentation describes a collection of C programs and shared subroutine libraries which implement a number of methods for the analysis of data observed on members of a pedigree structure. This pedigree structure may be large and/or complex. The implementation is directed primarily towards Markov chain Monte Carlo methods (Hastings, 1970; Thompson and Wijsman, 1990), but incorporates other tools for statistical genetic analysis. For example, EM algorithms (Dempster et al., 1977) are widely used in genetic analysis, and the non-Monte Carlo EM methods of Thompson and Shaw (1990, 1992), are included as a separate but related tool. Additionally, *peeling* (Elston and Stewart, 1971; Cannings et al., 1978; Hasstedt and Cartwright, 1979) is used as a tool for likelihood evaluation within the Monte Carlo algorithms. Peeling routines (for polygenic peeling and for major-gene peeling) and other likelihood evaluation routines are included as a part of one of the subroutine libraries.

More recent additions to the programs include the implementation of Metropolis coupled Markov chains (Geyer, 1991a), Monte Carlo mixtures (Geyer, 1991b), heated chains and other methods for addressing the problem of multiallelic markers in linkage analysis (Lin, 1993), and an alternative Markov chain Monte Carlo approach to autozygosity mapping.

This documentation can not provide details of the methods implemented; readers will need to refer to the cited papers if unfamiliar with the objectives and background of a main program. Conversely, the bibliography can not cover all current methodological developments in Monte Carlo genetic analysis; references are given only to the methods implemented here.

1.2 Organization

All libraries and programs are currently in /usr/users/pedpack/gibbs on the DECstation3100s. Within the GIBBS directory are main program directories, and subroutine library directories. The key method of installing and running programs is through Makefiles (see section 3). If the pathname to the gibbs directory is changed, then changes to the GIBBS parameter must be made in any Makefiles which are in program directories. The first line of most Makefiles is:

GIBBS = /usr/users/pedpack/gibbs

so if this pathname is changed, this line must be modified appropriately. The Makefiles that will need to be changed are those that are in program directories: Clip, Gibbs, MIXED, and PolyEM, McGibbs, LINK_LIN and Autozyg.

The subroutine libraries are Nghds, Quant, Mendel_Gibbs, McLib and Rans. These libraries are shared by the main programs, but not all main programs use all the libraries (figure 1). The key libraries are Rans (which contains random number and timing routines) and Nghds (which contains routines for manipulating the basic pedigree structure). Each of the other subroutine

libraries relates to specific program directories, each each itself requires Nghds. Most of the main program directories additionally require Rans. The libraries are also installed using Makefiles, the key command available being "make install".

The structure of the software is such that only one copy of subroutine libraries should be required on any machine. These should be installed and maintained by one individual responsible for the software. Source code for the libraries should not be copied or made generally available. On the other hand, individuals undertaking diferent types of analysis, fitting different types of model, or extending programs to handle different types of data, may need to modify main programs and immediately associated subroutines. Thus separate researchers may have their own copies of main program directories, modifying the routines in these program directories as required, but retaining the use of the shared installed libraries. This structure has proven useful in maintaining some coherence among the programs developed by different students and researchers, and is very strongly recommended in any installation of this software.

Test data sets are used to check correct installation of the programs. These are located in $(GIBBS)/Data/Test. Details are given later in the documentation.

1.3 Continuing development and testing

In addition to installation on a DEC3100, these programs have been installed on a SUN SPARCstation2 (St. Louis), and on a VAXstation running VMS (Seattle; Medical Genetics). Recently, a version has been installed on SUN SPARC2's in the Departments of Statistics and Biostatistics, University of Washington, for use and testing by students. Parts of the software have been ported to a DEC Alpha 300, for testing purposes. Generally, the goal is portability to any Unix workstation environment, although some programs may demand the availability of specific software (e.g. cmlib). Additionally, we are seeking collaborators who will test our programs in a non-development, non-expert, environment.

This documentation can describe only the programs available at a given point in time. The set of programs is under continuing development, and as new programs are added it is sometimes necessary to change details of earlier programs to maintain compatibility. When libraries are expanded, or the capabilities of a method or program are expanded, back-compatibility is checked for. Thus the same text data sets are used, and old and new programs must produce the same results.

2 Getting Started

This and the following section describe the installation of the software, written primarily for the person who will install and manage this package. The reader who is more interested in what the programs do, in terms of the methods and models implemented, may wish to skim this section at first reading, or to proceed immendiately to section 4, where an overview of the programs is given.

Before attempting to install and run the genetics programs, it will be helpful to orient yourself with the directory setup. The description of the program stuctures and interrelationships will assume a basic familiarity with Unix. (For example, *cd* will be used to refer to changing directory.) You should not attempt to implement this software on (for example) a VMS system, unless suffiently expert to translate the terminology and make any necessary changes to Makefiles.

The directories Nghds, Mendel_Gibbs, Quant, McLib and Rans are subroutine library directories. Quant also contains a subset of routines Peelie, which are partially separated. The header files for these libraries are in the Headers directory. The directories Clip, Gibbs, MIXED, Mc_Gibbs and PolyEM are main program directories. Data/Test is a subdirectory which contains the Test data.

This Getting Started section will refer to other sections; if installing and testing a particular set of programs, it may be wise to consult the section in which details of that program, input files, data formats etc., are described.

Please read through section 2 in its entirety before beginning.

2.1 Step One

Each subroutine library and main program directory contains a Makefile, as does the main directory (see Makefiles; section 3). The first change that needs to be made is to the first line of all *main program* Makefiles, which reads:

GIBBS = /usr/users/pedpack/gibbs

If this pathname is changed, this line must be modified appropriately in each main program Makefile.

2.2 Step Two

The main Makefile (in the $(GIBBS) directory) can be used to install all of the libraries, and to test the output of the programs using the test data against the gold standard output also provided for most main programs. The alternative to using the main Makefile to install the libraries and to test the output of the programs against their gold standards is to cd to each library and program directory, and run the appropriate Makefile commands using the Makefile within each subdirectory. Directions for the main Makefile will be detailed below. Also, please refer to the Makefiles section which has an extensive list of commands for each type of Makefile (main, program, and library).

To install the libraries and test the output of the main programs against the gold standard, type "make" while in the main $(GIBBS) directory. This will also lint the main programs and the library subroutines. To see the results, cd to each main program directory. The differences (using the diff command) between the output of the program and the gold standard will be in a file called diff.*program*, where *program* is the name of a main program. If the differences are small (such as the time and date stamp, rounding differences), there is no problem. The output of lint can be seen by cd'ing to each main program directory and library directory. In the main program directories, the output is sent to lint.*program*. In the library directories, the output is sent to lint.out. For more information on the Makefile commands, see the Makefile section. For more information on the gold standard and the test data, see the detailed descriptions of the main program directories that follow. *If you wish to focus on only one group of programs (such as the PolyEM programs) , you may also skip to the section on the PolyEM programs and follow the "Testing Upon Installation" procedure.*

2.3 Step Three

Once you have verified that the output of the main programs does indeed match that of the gold standard, you are ready to set up your own data files. The proper format of the data is described in the Data Format section 13, and in the detailed descriptions of the main program directories.

Assume that you have a data file ready to be used as input into a program. Note that the programs are also most conveniently run through the commands in each program-directory Makefile. Thus, it is now necessary to change the lines of the program-directory Makefiles which which provide the name of the data files, because these are still pointing to the test datasets.

Each main program Makefile (except Clip) has a line near the top which reads:

DATA = $(GIBBS)/Data/Test/etc...

This line needs to be modified to point to the new data directory in which data for the particular group of main programs will be kept. For instance, if you create a directory $(GIBBS)/Data/GAW to contain data for your GAW project, and if the data sets for running the PolyEM programs on these data additionally have the prefex *polyem*, then you could define:

DATA = $(GIBBS)/Data/GAW/polyem

in the Makefile in your PolyEM directory.

Next, wherever there is a statement that reads "*program*.out:" in a program-directory Makefile, the group of lines associated with it needs to be modified so that the input reflects the FULL pathname of the relevant data files. For example, in the PolyEM Makefile, there is the following group of lines associated with "nogenes.out :":

```
nogenes.out : nogenes $(DATA)_bivar45
        nogenes $(DATA)_bivar45 > nogenes.out1 2>&1
```

The full pathname for the test dataset is $(DATA)_bivar45, or more explicitly, with the above definitions:

/usr/users/pedpack/gibbs/Data/GAW/polyem_bivar45

The $(DATA) portion of this should have been modified by you (see above). The _bivar45 portion must be replaced to reflect the suffix of the dataset which will be used for the nogenes program.

2.4 Step Four

The next step is to run the programs using the Makefiles and your datasets. You have modified the Makefiles so that the programs will read your datasets as the input, so the only remaining step is to understand the commands that the main program Makefiles use. These are detailed in the Program Makefiles section 3. The command to load and run a main program called *program* is normally "make *program*.out", which will send the output to *program*.out1.

2.5 Step Five

In order to understand the output of the programs, and to learn more about data input and commands unique to certain program Makefiles, please refer to the detailed sections for each group of main programs which follow.

3 Makefiles

It is assumed that the person responsible for installing and maintaining these programs already has some familiarity with Makefiles and their use. Other users, especially if involved in developing or modifying programs, may like to consult documentation; a useful guide is Oram and Talbott (1993?).

There is a main Makefile in the main $(GIBBS) directory, and also there are Makefiles in each program/library subdirectory.

3.1 main Makefile

In the Main directory $(GIBBS), there is a Makefile which will do several tasks like installing all libraries, linting all programs and libraries, and loading all main programs and showing differences in .out files. The latter is especially useful upon first installation of the programs. Successful installation can be checked by using the supplied test data and this Makefile to show differences

between your output and the "gold standard".
Note: cd stands for change directory or set default directory.

Make Commands	**Result**
make	does everything described below
make diffg	cd's to Gibbs directory, removes all object code (.o files), loads the main program gibbs_test, runs the executable, and then uses the diff command to show any differences between the output in $(GIBBS)/Gibbs/Out1 and the gold standard output in $(GIBBS)/Gibbs/Out2. Output is sent to diff.gibbs_test.
make diffb	cd's to PolyEM directory, removes all object code (.o files), loads the main programs (bivar, univar, nogenes, and multivar), runs the executables, and then uses the diff command to show any differences between *program*.out1 and *program*.out2 (or *program*.out*) for each main program. Output is sent to diff.*program*.
make diffm	cd's to MIXED directory, removes all object code (.o files), loads the main programs (mixd and linkreg), runs the executables, and then uses the diff command to show any differences between *program*.out1 and *program*.out2 (or *program*.out*) for each main program. Output is sent to diff.*program*.
make clean	cd's to each main program directory (Clip, Gibbs, PolyEM, and MIXED), and removes all executables and object code (.o files).
make ultraclean	makes clean as explained above. Then cd's to each library directory (Mendel_Gibbs, Nghds, Quant, and Rans) and removes all compiled programs (.o files) and lib*libraryname*.a and llib-l*libraryname*.ln files, where *libraryname* is quant, mendel, nghds, or rans for each library. Removes all .a and .ln files from main directory.
make lint	cd's to each main program directory and lints each main program. Output is sent to lint.*program*, where *program* is the name of a main program (Fix Clipper: sends it to lint.out). Then, cd's to each library directory and lints subroutines. Output is sent to lint.out.
make libs	cd's to each library directory (Mendel_Gibbs, Nghds, Quant, and Rans) and installs the library (see Library Makefiles section for more information on installation).
make all	makes ultraclean, lint, and libs as explained above.

3.2 Program Makefiles

In each program directory (i.e. Clip, Gibbs, MIXED, and PolyEM), there is a Makefile. This Makefile will load any or all of the main programs in its directory, and will also execute them. It

will also do other tasks like cleaning the directory of object code (.o files). Details relevant for individual programs will be explained later with more information about each group of programs. General tasks that each of the program Makefiles will do are explained here.

Make Commands	**Result**
make	does everything described below.
make all	loads all main programs in the directory
make clean_up	removes all executables in the directory
make clean	removes all object code (.o files) in the directory
make ultraclean	removes executables and all object code (.o files) in the directory
make *program*.out	loads *program* and runs it, sending output to *program*.out1. *program* is just the name of any main program.
make *program*.diff	loads and runs *program*, and uses the diff command to show any differences between *program*.out and *program*.out2 (or *program*.out*). This is useful for seeing if there are any differences between the "gold standard" and your output. Output is sent to diff.*program*.
make diff	does make *program*.diff for each main program. Output is sent to diff.*program* for each program.
make *program*.lint	lints a single program (named *program*), and sends output to lint.*program*.
make lint	lints all main programs and sends output to lint.*program* for each main program.

3.3 Library Makefiles

Each library directory (Mendel_Gibbs, Nghds, Quant, and Rans) has a Makefile. This Makefile can be used for installing the library, linting the subroutines, and other tasks. Commands common to each library Makefile are explained below:

make clean	removes all object code (.o files) in directory
make ultraclean	makes clean as described above, and removes lib*libraryname*.a and llib-l*libraryname*.ln files, where *libraryname* is quant, mendel, nghds, or rans for each library.
make lint	Lints all subroutines and sends output to lint.out. Moves all .ln files to main directory.
make lib*libraryname*.a	Removes lib*libraryname*.a, then compiles, loads, and "archives" the executable file lib*libraryname*.a
make install	makes lib*libraryname*.a as described above, does everything necessary to install the library, then moves lib*libraryname*.a up one directory for actual use.

4 The Main Programs - An Overview

4.1 PolyEM Programs

The PolyEM programs are used to test deterministic EM equations in the case of an additive model with observed covariates (*fixed (covariate) + random (genetic) + error*). The programs implement improved efficiencies in the EM algorithm as applied to data on extended pedigrees (Thompson and Shaw, 1990, 1992).

The main programs are listed in the following table.

Brief Description of PolyEM main programs

univar	test program for a univariate trait
bivar	analyses bivariate quantitative trait data
nogenes	modification of *bivar*, used in the case where there is no genetic variance
unibig	a version of univar for very large or looped pedigrees
multivar	extends *bivar* to oligivariate traits; still under development

4.2 MIXED Programs

The MIXED programs are *linkreg* and *mixd*. The *mixd* program fits the mixed model (major Mendelian genes + polygenic additive effects) while linkreg does combined linkage and segregation analysis with the mixed model. Both programs use Monte Carlo EM (Guo, S. W. and Thompson, E. A., 1991, 1992) to find preliminary model estimates, and then use Monte Carlo Likelihood Ratio Evaluation (Thompson and Guo, 1991) to explore the likelihood surface in a neighborhood of the preliminary parameter estimates, and to estimate likelihood ratios for alternative models.

Additionally this directory contains the development programs for the Monte Carlo Mixtures method of estimating log-likelihood differences between disparate hypotheses (Geyer, 1991b; Thompson et al. 1993; Thompson, 1993). These programs use the prefix "mixt" – for example the main program is *mixt_mixd*. Additionally this directory includes test programs for peeling computations and for random number generator seeds.

The main programs are listed in the following table.

Brief Description of MIXED main programs

mixd	MCEM and crude MCLR estimation for the mixed model
linkreg	MCEM for the mixed model with a linked marker
mixt_mixd	
mixt_test	
mixt_multichain	Monte Carlo mixtures as in Thompson (1994a)
mixt_penetrance	MC mixtures with the modified mixed model of Thompson (1994b)
seed_test	tests the getting of seeds routines in Rans
acvs_test	tests the autocovariance routine *my_acvs*
polypeel_test	tests the polygenic peeling routine in Quant
mjrpeel_test	tests the Mendelian peeling routine in Quant
test_peel_seq	tests data input and peeling sequence determination routine in Nghds

4.3 LINK_LIN programs

The work of Lin (1993) extends the earlier *linkreg* program for combined segregation and linkage analysis to the case of multiple alleles, using "heated samplers" and "coupled chains". The *linkreg* program in the directory LINK_LIN has the same functionality as the earlier *linkreg* in MIXED, and additionally can resolve more complex examples in which the marker loci have more than two alleles. The MCEM and the lod-score curve evaluation parts of the program are separated. There are thus two main programs; *linkreg* does the Monte Carlo EM, while *linkage* computes a LOD score curve for a given segregation model.

4.4 Autozyg programs

A rather different Monte Carlo likelihood evaluation method is provided by taking segregation indicators or gene-identity-by-descent indicators as the latent variables, rather than the genotypes. This approach may be useful in multilocus mapping on small pedigrees, and specifically is being developed for autozygosity mapping of rare recessive traits in small sets of related affected inbred individuals (Thompson, 1994c). The main program *autozyg* is under development.

When only a single inbred individual is observed, we have the classical autozygosity mapping of Smith (1953) and Lander and Botstein (1987). In this case an exact solution to the probabilities of all multilocus gene identity patterns of the single observed individual is readily obtained by the recursive algorithm of Thompson (1988). This earlier program has been integrated into the Autozyg programs and routines, under the name *autolink*. Further work on this continues.

4.5 Clip Program

This directory contains one stand-alone program to preprocess a pedigree. The *clipper* program in the Clip directory uses the pedigree input and neighbourhood routines to input a full pedigree, and then clips the pedigree, removing individuals who provide no information for genetic analysis. It clips both up and down.

The first command line argument is a quadruplets file (pedigree triplets (individual, father , mother) + sex), and the second argument is a file that contains the data. The output file clipper.out contains quintuplets that can be read into the main Gibbs programs (after the first three lines are deleted).

4.6 Gibbs Programs

The programs in this directory are those using the pedigree Gibbs sampler routines for Mendelian traits, to estimate posterior probabilities of genotypic configurations, conditional on observed data (Sheehan, 1990). The main programs are *gibbs_test*, *mc_test* and *mc_hops*. The first tests the original Gibbs sampling for Medelian traits on pedigrees, and the remaining two are test implementations of Metropolis coupled Markov chain methods (Geyer, 1991a). This library of programs is superceded by the McGibbs programs.

4.7 McGibbs Programs

These are improved and generalized versions of the Gibbs programs; this directory supercedes the Gibbs programs; it will run the same test examples, and in addition allows more complex examples to be run. The single main program is *mc_gibbs*; running of this program for different examples is achieved via commands *test*, *quick*, *hops*, and *hutt770* and *short_drop* in the Makefile. (See section 3, for information of Makefiles.) Additionally there is a command *hutt3315* which runs a larger Hutterite example pedigree, but the data for this example is not publically available. Also there are two small input and analysis programs *newtrait* and *analyze*.

5 Header Files and data structures

5.1 Header files and directory interelationships

One way to obtain an overview of the relationships among programs and libraries (see also figure 1), is to examine what header files are *included* in program files. The Headers subdirectory contains headers for the subroutine libraries. The files *mclib.h, mendel.h, nghds.h, quant.h,* and *rans.h* are headers for the respective libraries. The header *peelie.h* is a separate header file required by some routines in the Quant library. The library headers *mclib.h, mendel.h* and *quant.h* include *nghds.h*, and *nghds.h* includes the standard C header *stdio.h*. The Rans library header, *rans.h* includes *stdio.h* and also *time.h* which defines times.

The routines in each library typically include only the relevant library header; they never include other library headers. Thus, for example, almost every file in the Quant library includes *quant.h*. Some also include *peelie.h*; the Peelie routines may be regarded as a subset of the Quant library. A few routines that perform simple manipulations may include only a system header such as *stdio.h*. A routine in Quant would never include *mclib.h*! Since the majority of routines in Quant require basic mathematical functions (log(), exp() etc.), the system header *math.h* is included in *quant.h*.

A few routines in other libraries may also require mathematical functions; in this case the routine separately includes *math.h*. (See for example, the routine *normal.c* in the Rans library.)

Additionally, most main-program directories include one or more header files for the main programs of that directory. These headers may also include some subset of the library headers in the Headers directory. The PolyEM directory does not have a header file; the main program files in PolyEM includes *quant.h* and in some cases *peelie.h* and other variables are defined within the program files. More typically, a main program directory contains a header file *head.h* which includes one or more library header files, and which is included by the programs in that directory. MIXED has two header files; one named *head.h* and used by the *mixd* progran, and the other *lhead.h* (which includes the *head.h* and additional items used by the *linkreg* program. Header files are where most of the predefined constants, such as maximal array sizes are located.

5.2 Pedigree data structures

The key data structures of the software are also defined in the header files, and the most important of these is the pedigree neighborhoods structure defined in *nghds.h* (Sheehan, 1990). In the current definition an individual has been expanded to contain additional items that may not be required by all programs.

An individual is defined in *nghds.h* by:

```
typedef   struct _Ind {
          struct _Ind *pa,*ma;
          int self; int index;
          int component;
          struct _Marlist *marriages;
          int ntrips; int *trips;
          } Ind;
```

Thus an individual consists of pointers to two other individuals, his father *pa* and mother *ma*, and an integer *self* which is the identifier of the individual. Another integer, *index*, gives the position of the individual in the internally held pedigree representation; in more recent programs this replaces the *position* array of Sheehan (1990). The integer *component* tells to which component pedigree of the data set the individual belongs. Finally, *marriages* is a pointer to a *Marlist* (see below); a linked list of marriages of the individual. The integers and integer pointers *ntrips* and *trips* are used only in a newer pedigree representation used by some of Geyers's programs. The pedigree is held as a set of triplets; *trips* identifies the ones the individual in a member of, either as a parent, or as the offspring.

A *Marlist* is a linked list of marriages. A marriage structure *Mar* consists of a pointer to an *Ind*, the *spouse* and a linked list of individuals (an *Indlist*) that are the children of the individual via that spouse. A pedigree *Component* is defined by an integer index and an *Indlist*, a linked list of the individuals that are its members. These individual and marriage structures are set up by *nghds*, the key routine in the Nghds library, and are fundamental to almost every program in this software.

6 The Subroutine Libraries

The subroutine libraries are Nghds, Quant, Mendel_Gibbs, McLib and Rans. Nghds and Rans are fundamental to almost all the programs, whereas the other libraries relate to a specific subset of the programs (figure 1). Only PolyEM which contains deterministic EM analysis programs does not include Rans. All the main programs includes Nghds, either directly (see Clip and Autozyg, for example) or through a subroutine library (see Quant or McLib, for example) that itself includes Nghds.

Note that the majority of program and library subroutine source code files contain not only a brief comment section stating their purpose, but also a brief history of their origin, and of any subsequent modifications. This is not only to acknowledge the original developer, but is also a record enabling us to track development of the package, and to track down the source of newly arising incompatibilities or difficulties.

6.1 Nghds

Nghds contains pedigree structure set-up and routines. The nghds array is a vector of individual structures (*Ind*); see section 5.2. The nghds array is the way in which the pedigree structure is stored and used in these programs.

Brief Description of Nghds Routines and Source Files

nghds.c	main routine (and its subroutines) to form pedigree neighborhoods, nghds
inbreeding.c kinship.c	routines to compute kinship and inbreeding coefficients
dump_data.c dump_nghds.c	output routines for checking whether input genetic data have been read in correctly, and whether neighborhoods have been correctly set up.
pedalloc.c	allocates space for pedigree structure
position.c	indexes a position of an individual specified by "name" ID; the index item of the nghds structure now provides alternative access to this information.

intmax.c	utility to find the maximum of a list of integers.
input_ped.c	core input routine that reads pedigree triplets, individual covariates (ints) and data variables (reals/ints).
components.c	routine to determine the disjoint components of a pedigree and set up structures enabling sampling or analysis to be on a component-by-component basis.
peel_seq.c	routine to determine peeling sequences for zeroloop pedigree (see section 12.2).

6.2 Quant

Quant contains routines for analysis of quantitative traits, including polygenic peeling routines (Thompson and Shaw, 1990, 1992). Also currently included are routines for evaluation of likelihoods for major gene and polygenic models on zeroloop pedigrees, using peeling algorithms.

Brief Description of Quant routines

input_data.c	input routine (calls input_ped() in Nghds)
cheigen.c	eigenvalue routines
henderson.c	routine to compute A^{-1} via Henderson's algorithm
reduce.c	routine to reduce correlation matrix A
out_fixed.c	part of the EM algorithm routines for covariate effects
innerprod.c invert_mat.c quarz.c	miscellaneous bits of matrix computation

Brief Description of likelihood evaluation routines

peel_addi.c	likelihood evaluation routines by polygenic peeling method
mjrpeel.c	likelihood evaluation routines by major-gene peeling for one or two loci
likval_addi.c	likelihood evaluation routines by methods other than peeling
old_linkpeel.c	major-gene peeling for linked loci for use with the original linkreg program only

6.3 Mendel_Gibbs

Mendel_Gibbs contains routines for Mendelian Gibbs, the sampling of Mendelian genes. This is the library used by the *Gibbs* main programs. The routines mc_menalloc.c and menalloc.c are used for allocating space for genetic structures.

6.4 McLib

McLib is the library used by the McGibbs main programs. Thus, as McGibbs supercedes Gibbs, so also McLib replaces Mendel_Gibbs.

6.5 The Rans subroutine library

The *Rans* library contains routines for random number generation, and also routines for timing test programs, and date-stamping output.

6.5.1 Introduction

Until 1992, the Gibbs sampling Pedigree prgrams used random number generators available on the DEC3100 on which the programs were developed. This proved unsatisfactory, both because different programs (by different authors) were using different random number generators, and because some level of transportability to other machines and systems became necessary.

In Spring 1992, E. A. Thompson implemented the **Rans** subroutine library, using one of the publically available random number generators as a source, modifying routines in other program directories and transferring them to the new library where appropriate, and adding new routines for retrieving and storing seed values for the random number generator.

6.5.2 The basic rans() routine, and get_seeds() and put_seeds()

The **Rans** subroutine library includes the uniform random number generator *rans*() of Marsaglia et al. (1973). This is generally accepted to be a better random number generator than the standard system uniform random number generator (also sometimes called *rans*(); do not confuse them!!) and than other Unix random number generators such as *drand*48(). It has worked well in tests, but any pseudo-random number generator should be treated with caution in any very extensive Monte Carlo study.

It is included with these programs primarily to enable users to obtain identical check results on the test data sets.

The routine requires a Random Seed consisting of two integers; there is a default value, or alternatively the routine *random*() will provide a random initialisation. However, we have found it preferable to read the seed values from a file ".Random.seed". If desired, the default values may be placed in this file. The routine "get_seeds()" initialises the random number generator using these seed values. The routine "put_seeds()" will put current values back to the file for a future run. If you do not wish to change the values in .Random.seed you should either delete calls to *put_seeds*() from the programs, or make the file *.Random.seed* unwritable. (See the Test Program section for further details.)

6.5.3 Other routines in the Rans library

Rans also includes routines for timing and date-stamping program runs, such as *stamp_time()*.

Additionally, there are other routines involving random numbers. These include *normal()*, which generates Gaussian random variables, using the well known polar method (Knuth, 1981; P.117), *rand_int()* which generates a random integer in a given range, and *permute()* which provides a random permutation of a list. These routines are used as tools by many of the main programs.

Brief Description of Rans routines

normal.c	generates Normal (μ, σ^2) random variables
rans.c	usual random number generator, uniform [0,1).
rans_.c	rans_.c is the Fortran equivalent of rans.c.

seed.c	contains routines to get seeds, restore seeds, etc... The range of the random seed is 2 random integers where the first is odd and the second is nonzero
get_seeds()	reads two numbers from your file .Random.seed, outputs them to the head of your standard output, and uses them as seeds for rans().
put_seeds()	outputs the final values of the seeds, after rans() has updated them as many times as your program calls rans(), to the file .Random.seed (unless you have write-protected it).
permute.c	provides a permutation of a list.
rand_int.c	generates an integer uniformaly from a specified range.
stamptime.c	times the program (measures elapsed time), and date-stamps your output.

6.5.4 A test program

A main program using stamp_time(), put_seeds() and get_seeds() should look something like:

```
#include <stdio.h>

extern double rans();
extern void get_seeds(), put_seeds(), stamp_time();

main()

    { int i; long cseed,tseed;
        long t1, time();
        time(&t1);
        get_seeds(&cseed, &tseed);
        for (i=0; i < 5; i++) printf(" %f",rans());
              * this one line is your program *
        stamp_time(t1);
        put_seeds (cseed, tseed ); }
```

NOTES:
1) The one line in the middle is your whole main program – whatever you want to do, presumably involving calling rans().

2) The call to put_seeds() is deliberately placed AFTER the final stamp_time(), since if you write-protect your .Random.seed, put_seeds() will exit the program.
3) If .Random.seed file contains 12345 1073, then the output should be (modulo a different date/time):

```
12345 1073
0.229256 0.766875 0.683177 0.509191 0.874560
Total elapsed time: 0 hr(s) 0 min 1 sec
Date and time: Tue May 12 15:46:04 1992

Cannot open .Random.seed for write
```

4) Remember that in order to use the gibbs/Rans library you will need the appropriate LDFLAGS

```
cc -O -I. -I/usr/users/pedpack/gibbs/Headers -c test_prog.c
cc -L/usr/users/pedpack/gibbs -o test_prog test_prog.o -lrans -lm
```

5) If you do NOT want to update your seed for the next run, either write- protect your .Random.seed, or do not call put_seed().
6) But if you find you do want to run with same seed again, you can always restore your .Random.seed from the values printed at the head of your program output.
7) The first of the two integers in your .Random.seed must be odd, the second non-zero; if these are not satisfied get_seed() will tell you and exit the program.
8) The two numbers above; 12345 1073 are the default values. Putting these two numbers in your .Random.seed will give the same output as if you do not call any seed initialiser (such as random()).

7 PolyEM Programs

The PolyEM programs are used to test deterministic EM equations in the case of an additive model with observed covariates (*fixed (covariate) + random (genetic) + error*) for bivariate traits on large pedigrees.

The programs implement improved efficiencies in the EM algorithm as applied to data on extended pedigrees (Thompson and Shaw, 1990, 1992).

The PolyEM programs use the Nghds and Quant subroutine libraries. Additionally, some of the programs (*univar* and *multivar*) require routines for matrix manipulation (inverses, determinants). These are currently linked in through publicly available *cmlib* library routines.

7.1 The programs bivar, nogenes, and multivar

The *bivar* program does bivariate analysis of two traits in accordance with the methods presented by Thompson & Shaw (1992). The *bivar* program uses the full **A**-inverse, instead of the **A** for observed individuals, where **A** is the "numerator-relationship-matrix" (Henderson, 1966). The program is adapted for a large pedigree (up to 1000), and takes as input the components of a sparse **A**-inverse, of the **A**-matrix for all individuals, and also a set of eigenvalues of the **A**-matrix (hence also of **A**-inverse) for observed individuals only.

The *nogenes* program is a modification of *bivar*, used in the case of no genetic variance. It is convenient to have a separate simple program to compute this null-hypothesis likelihood.

The *multivar* program is an extension of *bivar* for joint analysis of several (say up to about 6) traits. It replaces the bivariate explicit inverses and determinants by manipulation of (small) matrices (Thompson and Shaw, 1992). The program is not yet operational.

7.2 The programs univar and unibig

The *univar* program analyzes a single trait. It was developed as a test program, illustrating the methods of Thomspon & Shaw (1990). Cases of up to 250-individual pedigree are allowed. Being restricted to relatively small pedigrees, three alternative methods of likelihood evaluation can be employed (and checked for equality!), and other alternative implementations of matrix manipulations, and of the EM equations can be compared.
The methods included in *univar* include:
1) likelihood evaluation by peeling (Elston and Stewart, 1971), by Thompson & Shaw (1990), and by brute force including massive matrix stuff.
2) EM equations considering either all pedigree individuals, or just observed,
3) quadratic form to use either **A**-observed or **A**-all-inverse.

The *unibig* program is a version of *univar* for large pedigrees, or for complex pedigrees that cannot be "peeled". The peeling is omitted, as other computations are done by the bext method for large pedigrees.

7.3 Testing Upon Installation

As described in the Getting Started section, it is recommended that you run the programs using the test data and check the output against the gold standard before attempting to run the programs on your own data. This can be done using the main Makefile (detailed in Getting Started section), or you may use the PolyEM Makefile. To use the PolyEM Makefile, follow the directions through Step One in the Getting Started section. Next, while in the PolyEM directory, type "make *program*.diff", for each *program* (*bivar*, *univar* and *nogenes*). This command will load and run *program*, then uses the *diff* command to show any differences between *program*.out1 and the gold standard. The gold standard in this case is *program*.out.2. The differences will be in a file called diff.*program*.

For example, if you type "make bivar.diff", this will load and run the program bivar and diff the output bivar.out1 against bivar.out.2, the gold standard. The differences will be in the file diff.bivar. If the differences are few, such as differences in the time and date stamp, or small differences in the rounding of the numbers, etc..., then the programs have been successfully installed and are ready to run with your datasets.

Alternatively, you check installation using the main Makefile. In the Getting Started section, there are instructions on how to test all programs at once against their gold standards.

The test datasets are found in the $(GIBBS)/Data/Test subdirectory. The test datasets are polyem_bivar45 for bivar and nogenes, and polyem_univar45 for univar. Univar also uses polyem_peelseq45 for the peeling routines. A 45-member pedigree is used (Thompson and Shaw, 1990) for each of these datasets. The test versions do only a few cycles/scans/iterations to enable code to be tested against standard output.

7.4 Input

The basic pedigree information and trait data for input to the PolyEM programs is described in the Data Format section. The fourth integer, t, on the first line, gives the number of quantitative traits. For the bivar and nogenes programs, $t = 2$. For the univar program, $t = 1$.

Additional lines must be appended to the pedigree and trait input, in the same input file. These lines will contain information on the number of fixed effects to be modelled, and the initial values of these and other parameters. The following table explains line by line the additional input that needs to be appended to the end of the data. This table is for the *bivar* program, but is very similar for *univar* and *nogenes*. Differences will be noted following the table.

Line #	Input	Description
1	$k,\ a_1 \ldots a_i,\ b_1 \ldots b_i$	k = number fixed effects, not including overall means a_i= 0 if ith effect is in 1st trait, 0 otherwise; $i = 1, \ldots, k$ b_i= 0 if ith effect is in 2nd trait, 0 otherwise; $i = 1, \ldots, k$
2	c d e	c = initial value of the residual variance of the 1st trait d = initial value of the residual covariance of 1st and 2nd traits e = initial value of the residual variance of 2nd traits
3	f g h	f = initial value of the additive variance of the 1st trait g = initial value of the additive covariance of 1st and 2nd traits h = initial value of the additive variance of 2nd trait
4	$k_1\ k_2$	k_j = mean value of jth trait The raw average is a good mean value to use here

5	m_{1_1} n_{1_1}	Let i_j stand for the jth level of the ith effect Let r = number of levels for the 1st effect Let s = number of levels for the 2nd effect, etc... m_{i_j} = starting values for jth level of ith fixed effect for trait 1 n_{i_j} = starting values for jth level of ith fixed effect for trait 2
⋮		
5 + r - 1	m_{1_r} n_{1_r}	
5 + r	m_{2_1} n_{2_1}	
⋮		
5 + r + s - 1	m_{2_s} n_{2_s}	m_{2_s} = starting value for sth level of 2nd fixed effect for trait 1 n_{2_s} = starting value for sth level of 2nd fixed effect for trait 2
⋮		And so on in this pattern if there are more than 2 fixed effects

In the case of no fixed effects ($k = 0$), there are no "a" or "b" terms in line 1, and only lines 1 through 4 are relevant. The same model, but with a slightly different iteration procedure, can be achieved by the line

3 0 0 0 0 0 0

which would indicate that there are 3 fixed effects, but that none is to be modelled in either of the two traits, but in this case the program would expect the starting values (0.0, presumably!), to be given. For example, the final lines of the test data set $(GIBBS)/Data/Test/polyem_bivar45 are

2	0	0	1	1	number and structure of fixed effects
1.10	-0.09	0.65			initial residual variance-covariance terms
0.037	-0.0017	0.0288			initial additive variance-covariance terms
0.065154	1.806705				initial values for overall means
0.0	-0.765888				first effect (sex) has two levels
0.0	0.583534				first level is female, second is male
0.0	-0.991605				second effect (generation) has 3 levels
0.0	-0.553063				
0.0	1.159199				

There are two classes of fixed effect; neither is to be modelled in the first trait, but both are to be included for the second. The number of levels of each effect is determined by the program; there are, in fact, two levels of the first effect (sex) and three of the second, but this need not be specified. Since the effects are not modelled in the first trait, the initial values are set to 0.0 – but any values could be entered – they are ignored. Note the values in this test data set are actually estimates from an earlier run, simply for convenience in testing. In practice, initial values will not normally be given to such accuracy! Note also that the input is free format, with values separated by spaces or newlines; the arrangement above is simply for convenience – only the sequence of items matters.

Initially, in exploring these programs, the recommended limit on the number of classes of fixed effects is 2, in addition to the overall mean. The constant FIXEFF is currently set to 3 in Head-

ers/quant.h; this means a maximum of 2 plus the overall mean.

Also, the recommended limit on the number of levels of fixed effects is 6. In the Headers/quant.h file, LEVELS is currently 16 for testing a particular data set with one effect at 15 levels, but this was a large data set!

Note there is also a constant MAX_FEP defined in Headers/quant.h (currently 20); this relates to different parameter sets for different models used by other programs (e.g. mixt_multichain in MIXED), not to effects and levels for any one model.

If more fixed effects or levels are needed, then FIXEFF and/or LEVELS can be redefined in Headers/quant.h. If this is done, the libraries need to be re-installed, and the executables would need to be remade. This is not recommended for the inexperienced user.

The previous table is for the *bivar* program. It is the same for the *nogenes* program except for line number 3. For *nogenes*, f g and h on line 3 will each be equal to zero, because nogenes is used in the case of no genetic variance.

The differences for the *univar* program are that there is only one trait (instead of two). Thus, the changes will be as follows:

Line 1: No "b" terms
Line 2: No "d" and "e" terms
Line 3: No "g" and "h" terms
Line 4: No k_2 term
Line 5++: No "n" terms

The final lines of the test data file $(GIBBS)/Data/Test/polyem_univar45 are

```
0
2.0 3.0
0.0
```

indicating simply that no fixed effects are to be modelled (line 1), initial residual and genetic variances 2.0 and 3.0 (lines 2 and 3, concatenated), and a zero intial value for the overall mean (line 4).

The univar program requires an additional input file, which is for the peeling routine. The separate file contains a peeling sequence which specifies the sequence of operations for an exact "peeling" computation of the likelihood on a pedigree without loops. Details of the format of a peeling sequence file are given in section 13.2.

7.5 Output

If the command format in the Makefile is used to run the programs, output of each program will be in a file called *program*.out1. The format of the output for the nogenes and the bivar program is identical. For univar, there are small differences which will be pointed out as disparities arise.

The output lines given as examples relate to the file bivar.out1 formed by running bivar on the test data set $(GIBBS)/Data/Test/polyem_bivar45 used as the example in the previous section.

The first line of the output is the first line of the input file as a check that it has been read in correctly. The second line is also a check which reports the number of individuals in each pedigree. Next, the number of levels for each fixed effect is reported. Then, for each trait, it reports the presence (=1) or absence (=0) of a fixed effect. For example, the output may have the following form:

```
trait   1   1   0   0
trait   2   1   1   1
```

The first column of numbers refers to the trait. The second column of numbers refers to the overall mean, and should always be a column of 1's because it is always present. The 3rd column of numbers refers to the first fixed effect. A zero in the first row implies that the first fixed effect is absent from the model for trait 1. A one in the second row implies that the first fixed effect is present in trait 2. Similarly, the 4th column of numbers refers to the second fixed effect. A zero in the first row and a one in the second row implies that the second fixed effect is absent in the first trait but present in the second trait. The output from univar will not have the line for trait 2 as there is only 1 trait.

Next, if there are fixed effects other than the overall mean, then the number of individuals observed at each level for each fixed effect is reported. For example, the output may have this form:

```
fixed effects class 1 : number observed in each category = 16 21
fixed effects class 2 : number observed in each category = 8 15 14
```

The 16 refers to the number of individuals who are observed to have the first type (level) of the first fixed effect. The 21 refers to the number of individuals who are observed to have the second level of the first fixed effect. Similarly, the 8, 15, and 14, refer to the number of individuals observed to have the first, second, and third level of the second fixed effect, respectively. Each row should add up to the same number, which in this case is 37, the number of individuals observed for both traits. This is less than the total of 45 pedigree members, reported at the top of the output due; 8 individuals are unobserved.

The next line reports the number observed and the number of fixed effects. Following that is a list of the eigenvalues of **A**-inverse, for the **A**-matrix for the observed individuals. The dimension of the matrix is the number of observed individuals (37 in this case). The determination of eigenvalues is as described by Thompson and Shaw (1990), but with improvements suggested by R. Thompson (pers. comm.) and F. Shaw (pers. comm.). A number of eigenvalues are immediately determinable; the remainder are determined from a reduced matrix. The number of immediate eigenvalues and reduced dimension is printed preceding the eigenvalues.

Next, the starting values for the parameters are printed with the current log-likelihood value. For bivar, they are listed in the following order. (The format is slightly different for univar, and is given later.)

current variance estimates
additive *estimate(s)*
residual *estimate(s)*
fixed effects for trait 1
0 *estimate*
1 *estimates*
2 *estimates*
fixed effects for trait 2
0 *estimate*
1 *estimates*
2 *estimates*
Current lhood value = *estimate*

The 0, 1, and 2 under the fixed effects portion correspond to the overall mean, first fixed effect, and second fixed effect, respectively. There may be more or less than 2 fixed effects, but 2 is the number in this example. The estimates that follow the 1 and the 2 under the fixed effects are the estimates for each level of the fixed effect. In this example, fixed effect 1 has two levels, and fixed effect 2 has three levels. Thus there are two estimates following each 1 and three estimates following each 2 in the table above. However, since effects 1 and 2 are not modelled in trait 1, the trait-1 fixed-effect estimates for effects 1 and 2 will be 0.0.

Following this, the values of the parameters and the log likelihood value is printed every *ITWT*th iteration. This continues until the *ITEM*th iteration, and then the *(ITEM+1)*th (final) iteration is printed. The constants ITWT and ITEM are defined at the head of main program files, *program*.c, and can be reset if desired. Currently, ITWT = 50 for bivar, and ITWT = 10 for univar. Currently, ITEM = 200 for bivar, and ITEM=100 for univar. Also, univar and bivar have a variable ITMO, which is the number of breeding-value iterations to do within each iteration. This is currently set at 20, but can be changed. The setting for ITWT is completely up to the user as it merely affects how much output one gets; an ITEM setting of 100 is generally sufficient for most problems; a setting of 20 for ITMO is necessary for the early EM steps (even more may be necessary for large problems).

7.5.1 Differences for the univar output

The univar program's output is similar to bivar's output up to the point of at which the initial parameter estimates are printed. At this point it varies slightly, as described below. First, it prints out the initial parameter values:

current estimates *estimate1 estimate2 estimate3*

where estimate1 is the mean value of the 1st (and only) trait, estimate2 is the initial value of the additive variance of the trait, and estimate3 is the initial value of the residual variance of the trait. Next, the program prints out the following every *i*th iteration:

current (*i*) variance estimates *estimate4 estimate5*
and fixed effects

0 *estimate*
1 *estimates*
2 *estimates*
current lhood value (a la Thompson & Shaw) = *estimate*
current lhood value (brute force computation) = *estimate*
current lhood value, by peeling = *estimate*

where *estimate4* and *estimate5* are the additive and residual variance estimates, respectively. The 0, 1, and 2 under the fixed effects portion correspond to the overall mean, first fixed effect, and second fixed effect, respectively. As for bivar, there may be more or less than 2 fixed effects. The estimates that follow the numbers 1 and the 2 under the fixed effects are the estimates for each level of the fixed effect, as in the bivar case. Lastly, the univar program prints out estimates for three versions of the log likelihood, obtained by the Thompson and Shaw (1990) method, brute force, and by peeling. These three values should always be equal to within small rounding errors. If the peeling value is different, there is probably an error in the peeling sequence file. If the Thompson and Shaw value is different, there may be problems in the eigenvalue determination.

8 MIXED Programs

8.1 Mixd and Linkreg

The main MIXED programs are *linkreg* and *mixd*; only these two have been widely tested, and have "gold-standard" test input and output. Mixd fits the mixed model (major Mendelian genes + polygenic additive effects) while linkreg does combined linkage and segregation analysis for the mixed model. Both programs use Monte Carlo EM (Guo and Thompson, 1991, 1992, 1994) to find preliminary model estimates, and then use Monte Carlo Likelihood Ratio Evaluation (Thompson and Guo, 1991) to explore the likelihood surface in a neighborhood of the preliminary parameter estimates, and to estimate likelihood ratios for alternative models. A good reference for this implementation of Monte Carlo EM methods is S. W. Guo's dissertation (1991). The MIXED programs both use the Nghds, Quant, and Rans subroutine libraries, as well as subroutines in the MIXED directory: trans_prob.o, strucmisc.o, counting.o, gene_effect.o, output.o, pop_gene.o, cond_p.o, and smpl_ga.o.

Mixd also uses the following subroutines which are currently kept in the MIXED directory: mixed_input.o, genedrop.o, adjust.o, and calgenefrq.o.

Linkreg also uses the following subroutines which are currently kept in the MIXED directory: link_input.o, lfrwdsmpl.o, lsegreg.o, lfounder.o, lrecomcount.o, lcomprecom.o, mcond_p.o, lcalgenefrq.o, lp_ratio.o, ldistpeel.o, lperiph.o, and lpoly_ratio.o.

8.2 Other MIXED programs

There are numerous other programs in the MIXED directory, all relating to the fitting of a mixed model for a single quantitative trait, and to the Monte Carlo estimation of likelihood ratios. These

are listed in the overview section 4.2.

Some are test programs for various components of the overall procedures, for example peeling-sequence determination (*test_peel_seq*), peeling routines (*mjrpeel_test* and *polypeel_test*), random number routines (*seed_test*) and batchmean autocovariance evaluations (*acvs_test*).

Others are programs implementing the method of Monte Carlo mixtures (Geyer, 1991b) for the estimation of relative likelihoods. These have the prefix *mixt_*. The program *mixt_multichain* does the basic mixtures of several independent chains, each at a predefined mixed model (Thompson, 1994a), while *mixt_penetrance* uses the fractional penetrance modified mixed models of Thompson (1994b). *mixt_mixd* and mixt_test are similar test and development programs for MC mixtures.

These programs have been tested primarily on the data for Genetic Analysis Workshop 8 (Thompson et al., 1993). There are no "gold-standard" output files distributed.

8.3 Testing Upon Installation

As described in the Getting Started section, it is recommended that you run the programs using the test data and check the output against the gold standard before attempting to run the programs on your own data. This can be done using the main Makefile (detailed in Getting Started section), or you may use the MIXED Makefile. To use the MIXED Makefile to do this, follow the directions through Step One in the Getting Started section. Next, while in the MIXED directory, type "make *program*.diff", for each *program* (mixd and linkreg). This command will load and run *program*, then uses the diff command to show any differences between *program*.out1 and the gold standard. The gold standard in this case is *program*.out2. The differences will be in a file called diff.*program*. For example, if you type "make mixd.diff", this will load and run the program mixd and diff the output mixd.out1 against mixd.out2, the gold standard. The differences will be in the file diff.mixd. If the differences are few, such as differences in the time and date stamp, or small differences in the rounding of the numbers, etc..., then the programs have been successfully installed and are ready to run with your datasets.

Alternatively, you can do this using the main Makefile. In the Getting Started section, there are instructions on how to test all programs at once against their gold standards.

The test datasets are found in the $(GIBBS)/Data/Test subdirectory. A 59-member pedigree is used for each of these datasets (Leppert et al., 1986). The test datasets are *mixed_mixd* for *mixd*, and *mixed_linkreg* for *linkreg*. Both programs also use the peeling sequence file *mixed_peelseq* for the peeling routines (see section 12.2). The test versions do only a few cycles/scans/iterations to enable code to be tested against standard output.

8.4 Input

The basic input of both mixd and linkreg is described in the Data Format section. The number t as defined in the Data Format section is 1 for each program, where the tth column is the trait phenotype for each individual. The main difference in the basic input for linkreg is that there is an extra column of integers (the sth column, where s is defined in Data Format section) for linkreg,

where this sth column of integers is the information on the marker phenotype of individuals. Thus, the first line of the mixd and the linkreg programs will have t=1, and if s=n for the mixd program, then s=n+1 for the linkreg program due to the marker phenotype data.

For both programs, additional lines need to be appended to the basic data structure mentioned above, and detailed in the Data Format section. These additional lines contain information on number of fixed effects and initial parameter values. The following table explains line by line the additional input which needs to be appended to the end of the data for both mixd and linkreg.

Line #	Input	Description
1	i $a_1 \ldots a_k$	k = number of fixed effects a_i= 0 or 1 as ith effect is not or is in trait $i = 1, \ldots, k$ NOTE: If $k = 0$, then there are no a terms. Alternatively, you may write this line as: 2 0 0 for instance, meaning two fixed effects but neither is in 1st trait.
2	j	j = number of genotypes at trait locus (j=3 is necessary, as the program assumes 3 trait genotypes)

3	a b c d e	parameters for the Monte Carlo estimation
		a = # scans/realization
		b = # EM iterations
		c = # realizations/EM iteration
		d = # scans of burn-in before initial sampling
		e = # final EM steps used to give final estimates
		(See Guo and Thompson, 1992; see also figure 2)

4	p r σ_a^2 σ_d^2 σ_e^2	initial parameter values p = the major gene allele frequency r = the parameter that measures recombination between trait and marker locus (The linkreg program has the additional feature of a "marker locus" linked to the trait locus). r is FOR THE LINKREG PROGRAM ONLY (for mixd leave this BLANK!) σ_a^2 = the additive genetic variance σ_d^2 = the dominance variance. Note: This must be set equal to zero because it is not clear that the program works with non-zero dominance components σ_e^2 = the residual (error) variance
5	f	f = initial parameter value for overall mean effect
6	$g_1 \ldots g_j$	initial parameter values for the j genotypic effects Note: initial parameter values for fixed effects are set by program
7	mixd.lik	file name for the output of likelihood values for future analysis (this should be link.lik for linkreg program).
8	mixd.est	file name for the output of estimate values for future analysis (this should be link.est for linkreg program).

In the Headers directory, the *quant.h* file defines FIXEFF and LEVELS to be the maximum number of fixed effects (including the mean) and levels of fixed effects, respectively. The recommended limit on the number of fixed effects is three, including the mean; FIXEFF =3. Also, the recommended limit on the number of levels of fixed effects is six, although LEVELS is set to 16 (see section 7.4). In theory, the programs should work if FIXEFF and LEVELS are reset at higher levels. If this is done, the libraries need to be re-installed, and the executables would need to be remade.

The program first performs d pedigree scans under an initial model, without sampling. Then every a scans the realised configuration of major and polygenotypes is sampled, and a total of c configurations are sampled in order to update the parameters estimates. This last stage (without any initial d scans) is repeated b times – i.e. b MCEM steps. For the final e EM steps the MC sample size c is multiplied by 4 (doubling the precision), and the "final parameter estimates" are the average values over these e steps. For the final MCLR estimate a sample of size $8c$ is taken (again at a spacing of a pedigree scans) at these final parameter estimates. Recommended settings for $a - e$ above are as follows: $a = 2 - 5$ is usually good, a should be small; $b = 50$–100 is usually a sufficient number of EM iterations, although sometimes 200 EM steps are needed; $c =$ 100 minimum is recommended, ideally c should be very large but the tradeoff is computer time; d is almost irrelevant, in most examples; $e =$ a number which makes the ratio of e/b in the 20-30% range (i.e. if b=100, then e = 30 is good).

The *mixd* and *linkreg* programs also require a separate input file for the peeling routine. The mixd program uses polygenic peeling (*polypeel* in Quant) as a part of its Monte Carlo evaluation of likelihood ratios between alternative models. Linkreg uses a different peeling program, (*old_linkpeel* in Quant), but both programs use the same peeling sequence file. The peeling sequence specifies the sequence of operations for an exact "peeling" computation of the likelihood on a pedigree without loops (see section 12.2). The test peeling sequence for *mixd* and *linkreg* is in the file $(GIBBS)/Data/Test/mixed_peelseq. The current version of *mixd* estimates likelihood ratios for the final model relative to a major gene model with the same major gene effects. The likelihood for that major gene model is computed exactly (when possible) using the routine *mjrpeel* in Quant. A likelihood for a pure environmental model is always possible, and is computed, and can be used as a base-point for comparisons when other exact computations are infeasible.

8.5 Output

The output for the *mixd* program will be in a file called *mixd.out1*, and the output for *linkreg* will be in a file called *link.out1*. The first line of the output will be a string of numbers which are components of a random seed. This is printed out as a check, but also so that it is possible to rerun from some seed to print out more information if a certain run does something particularly interesting. These numbers are read from the file *.Random.seed* by the *get_seeds* routine. The *.Random.seed* file is in the MIXED directory, and *get_seeds* is in the *seeds.c* file located in the Rans library.

The first line of the linkreg output and the next line of the mixd output that is printed out is the first line of the data input file as a check that it has been read in correctly. The following line

is also a check, which reads "namestring = m" where m is the maximum number used as an ID number for the individuals.

Next, the number of levels for each fixed effect (if any) is printed. Following this, the presence (=1) or absence (=0) of fixed effects from the trait is reported. For example, the output may have the following form:

trait 1 1 0 1

The first number refers to the trait, which is 1 (mixd and linkreg only handle single trait cases). The second number refers to the overall mean, and should always be a 1 because it is always present. The remaining numbers are for the fixed effects. If there are no fixed effects, there will be no more numbers. In this example, there are two fixed effects. The third number is a zero, which means that the first fixed effect is absent from the trait. The fourth number is a one, which signifies that the second fixed effect is present in the trait.

For each fixed effect, the number of individuals observed in each category (level) is then reported. (In the case of 0 fixed effects specified by 0 on line 1, this table will not be printed. However, if line 1 reads 2 0 0 (2 fixed effects but neither in the trait), then this table will be given.) For example, the output may have this form:

fixed effects class 1 : number observed in each category 19 28
fixed effects class 2 : number observed in each category 1 13 33

The 19 refers to the number of individuals who are observed to have the first type (level) of the first fixed effect. The 28 refers to the number of individuals who are observed to have the second level of the first fixed effect. Similarly, the numbers 1, 13, and 33 refer to the number of individuals observed to have the first, second, and third type (level) of the second fixed effect, respectively. Each row should add up to the same number, which is 47 in this case; there are 47 observed individuals. Due to missing values, this number is less than the pedigree size, 59, reported at the top of the output.

The output then reports the type of model which is being estimated. For the mixd program, this will read "Mixed model with additive polygenic component," and for the linkreg program it will be "Linkregation analysis." Next, the total number of individuals in the pedigree and the total number observed are reported.

Now, the parameters for the Monte Carlo estimation are printed out:

Number of scans per sampled realisation: a
Total number of EM iterations: b
Number of realisations per EM iteration: c
Number of realisations used to evaluate ln likelihood: $8c$
Number of scans for initial burn-in: d
EM-steps for final averaging: e

These are simply the parameters a-e from line 3 in the Input section, and the number of realizations used to evaluate the log-likelihood is $8c$ by default; see previous section.

For the linkreg program, the following two lines are printed out:

Number of genotypes at marker locus: #
Gene frequency at marker locus: #

These numbers are calculated by the program and printed as a check. For both mixd and linkreg, the number of founders in the pedigree is also reported. The number of founders is the number of individuals at the "top" of the pedigree; those without parents (mother=father=0). This number is printed out as a check.

Next, the starting values are printed. They are printed as p, r, siga, sigd, and sige, which correspond to p, r, σ_a^2, σ_d^2, and σ_e^2 of the fourth line of additional input for the main data file (see Input section). Note that there is no r term for the mixd program. For both of these programs, sigd = 0 ($\sigma_d^2 = 0$); although component routines work, the programs as a whole have not been tested for a mixed model with non-zero dominance variance. After these initial parameter values are printed, it lists the "Major gene effects." For each of the "j" (line 2, Input section) genotypes, the initial values of additive effects of these genotypes are printed. Next, the starting values fixed effects are printed in the following form:

Fixed effects
0 *starting value*
1 *starting values*
2 *starting values*

This is an example for the case of two fixed effects. The 0th fixed effect is the overall mean. Only the overall mean was given a starting value by the user; the starting values for each level of the fixed effects $1, 2, \ldots$ are set by the program.

From this point on, the mixd and linkreg output differ more substantially. Thus, further discussion about each program's output will be in separate sections.

Mixd Output Continued

The program now prints out every ith EM step (beginning with 0). Near the beginning of the mixd.c file, there is a line which currently reads reads "define IWRT 5." This means that every 5th EM step will be printed out. To print more or less of the EM steps, simply change this number. For each step, the following is printed:

EM step i
p=*estimate* siga=*estimate* sigd=*estimate* sige=*estimate*
Major gene effects = *estimates*
Fixed effects
0 *estimate*
1 *estimates*
2 *estimates*

This example is for the case of two fixed effects. The estimates that follow the 1 and 2 under the fixed effects are the estimates for each level of the fixed effect. For example, if fixed effect 1 has two levels, and fixed effect 2 has three levels, then there will be two estimates following the 1 and three estimates following the 2. Fixed effect 0 is the overall mean.

The final estimates for likelihood surface exploration are then printed in the same format described in the previous paragraph. Now, the following is printed:

> Final major gene adjustment: adj
> population-adjusted overall mean: v
> and major gene effects: $w_i, i = 0, 1, 2$
> Basepoint environmental log-likelihood: ll

The major gene adjustment centers the major gene effects, so that effects are relative to a population mean under the model. Subtracting the final major gene adjustment adj from the population-adjusted overall mean will yield the overall mean (fixed effect 0) given in the final estimates ($v - adj$ = final estimate for overall mean). Adding the final major gene adjustment to the major gene effects will yield the major gene effects reported in the final estimates ($w_i + adj$ = final estimates for major gene effects). Of course, this final centering does not change the estimated model; it is simply a convenient unique specification.

The number of realizations for likelihood evaluation is the next item printed. This number is $8c$ (where c is the number of realization per EM iteration). Then, the log likelihood for the environmental model is reported, which should be equivalent to the basepoint environmental log-likelihood.

In the early versions of the program, the likelihood ratios are reported for the mixed model versus several other models: environmental, dominant, recessive, model with environmental component only, model with additive component only, model with major gene + error, and model with major gene + additive polygenes. Also, the log likelihood ratios for MLE neighboring points are reported. These show log likelihood ratios locally against alternative mixed models. The particular ones given in the gold standard output (mixd.out2) show what happens when siga (σ_a^2) and sige (σ_e^2) are altered a bit from the MLE's.

However, these lines are commented out of the current source code, and only a basepoint (exact) environmental log-likelihood, an (exact) major-gene model log-likelihood, and a Monte Carlo estimate of the log-likelihood of the mixed model relative to the major gene model.

Linkreg Output Continued

After the starting values are printed for the linkreg program, the number of samples and number of cycles are reported. The number of realizations used in the log-likelihood evaluation is the number of "samples," and the number of scans between each realizations is the number of "cycles."

Next, the final estimates are reported in the following form:

> p= *estimate* r=*estimate* siga=*estimate* sigd=*estimate* sige=*estimate*
> Major gene effects *estimates*

Fixed effects
0 *estimate*
1 *estimates*
2 *estimates*

This example is for the case of two fixed effects. The estimates that follow the 1 and 2 under the fixed effects are the estimates for each level of the fixed effect. For example, if fixed effect 1 has two levels, and fixed effect 2 has three levels, then there will be two estimates following the 1 and three estimates following the 2. Fixed effect 0 is the overall mean.

The program then prints the estimated log-likelihood ratio for MLE vs. other models. The format for each ratio is:

p = # r = # siga = # sige = #
mu_1 = # mu_2 = # mu_3 = #
log_likelihood ratio: #

The values of p, r, siga, and sige are the values of p, r, σ_a^2, and σ_e^2 for the alternative model. No sigd is given because this is always zero. The values of $mu_i, i = 1, 2, 3$ are additive effects of the three trait genotypes. The resultant log-likelihood ratio is then reported. The models which are chosen for comparison are obtained by varying p, siga, sige, and the mu's (additive effects for the genotypes) somewhat from the MLE's.

9 LINK_LIN Programs – Further development of linkreg

(Information provided by Shili Lin)

9.1 Linkreg

The *linkreg* program has now been further developed by Shili Lin using the ideas in Lin (1993). The new program version is in the directory LINK_LIN. *Linkreg* now does combined segregation and linkage analysis with a marker with an arbitrary number of alleles. (The version of *linkreg* in MIXED only handles diallelic markers.) The companion chain method (Lin et al., 1993) is implemented to ensure irreducibility of the Markov chain sampler. In this method, several samplers with different genetic models are coupled via a Metropolis switching scheme (Geyer, 1991a). A model must therefore be specified for each chain. The closed classes of communicating states under the original model are referred to as *islands*. The modified models provide irreducible chains, and the Mtropolis switching of genotype configurations between chains ensures ireducibility of the system, and in particular permits the realizations for the original model to jump between islands.

The previous MIXED and Quant subroutines which have been modified are: *genedrop.c, input_data.c, mcond_p.c, lrecomcount.c, linkreg.c mjrpeel.c, link_input.c, lcalgenefrq.c, lfrwdsmpl.c, ldistpeel.c, lperiph.c, segreg.c, smpl_ga.c.* The new subroutines are: *input_mk.c, founder_gfrq.c, llegal_conf.c, lfull_scan.c, mkalloc.c, auxcond_p.c, auxcond_z.p.*

There is also a new input file in addition to the data file (section 12.1) and peeling sequence file (section 12.2). This additional input file contains information about the marker locus and specifies the penetrance matrices required for each model used in one of the chains; details are given in section 12.3. The data input file (12.1) is also modified to include indicators to accommodate individual specific penetrance for the companion chains.

9.2 Linkage

Linkage is a program which does two-point Monte Carlo linkage analysis. The most recent version has been developed to permit estimation of multipoint *lod* scores, but is not yet fully tested. The *linkage* program requires three data files, the pedigree data file (section 12.1), the peeling file (section 12.2) and the marker data file (section 12.3). The input files have been modified to permit use of several markers; details are given in sections 12.1 and 12.3. The output of the program are LOD scores at 0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45 and 0.5.

The program is currently implemented to handle up to six markers with an arbitrary number of alleles. (Note also that the number of genotypes at trait locus is no longer restricted to 3.) The header file "head.h" has been modified to use dynamic memory allocation. However, the limitation of six loci is due to problems in allocating space (using *malloc*); these difficulties will be resolved as the program develops further. The data at each marker locus must be preprocessed to find all the *islands* (using a separate program) so that appropriate penetrances and companion chains can be constructed (Lin et al. 1994).

Most of the subroutines for two-point analysis have been modified to accommodate multiple markers. Subroutines which have been modified are *input_ped, input_mk, lcalgenfrq, lfrwdsmpl, lfull_scan, link_input, mcond_p, mkalloc, segreg* and *smpl_ga.*

9.3 Linkage Output

The output before "Linkage Analysis" is the same as *linkreg* output which was previously described in section 8.5. Following "Linkage Analysis" is a line specifying the segregation model parameters for the trait locus. Using this trait model, lod scores at specified recombination fractions are estimated. The six parameters of the trait model are the (disease) allele frequency, the additive and environmental variances (σ_a^2 and σ_e^2), and the three genotypic means at the trait locus.

Here is an example of the output:

```
check 232 6 1
namestring = 706

Linkage Analysis

The Model:
0.044 550.0 503.0 314.0 314.0 173.0
```

r	log(L(r)/L(0.5))/log10
0.01	-13.358
0.05	-4.905
0.10	-2.041
0.15	-0.788
0.20	-0.170
0.25	0.115
0.30	0.330
0.35	0.375
0.40	0.222
0.45	0.081
0.50	0.000

The output file for the case of multiple markers is very similar; an example follows. The recombination between markers is assumed known; in this output file, there are just two markers with recombination frequency 0.22 between them. The *lod* scores are then given at recombination frequencies r of the trait from the left-most marker.

```
check 9 8 0
namestring = 9

Linkage Analysis

2 markers with recombination vector between pairs of consecutive markers:
0.22

#scans for averaging=1000
```

r	log(L(r)/L(0.5))/log10
0.01	-1.288
0.05	-0.615
0.10	-0.330
0.15	-0.192
0.20	-0.113
0.25	-0.060
0.30	-0.026
0.35	-0.006
0.40	0.004
0.45	0.007
0.50	0.000

10 Clip Program

The "clipper" program in the Clip directory clips a pedigree, removing individuals who provide no information for genetic analysis. It clips both up and down.

The first command line argument is a quadruplets file (triplets + sex), and the second argument is a file that contains the data somewhere. The output file clipper.out contains the quintuplets that can be read into the main Gibbs programs (after the first three lines are deleted). Individuals are discarded by having their neighborhood indicators turned off so that they will be ignored in any estimation procedure.

The program clips down by "clipping" off the "loose" or uninformative individuals at the top of the pedigree. In the case of a single child of a founder couple with no other marriages involved, the child is turned into a founder and the neighborhood indicators are turned off for the parents. This is only done if there are no data on the parents.

The program clips up by "clipping" off the "loose" or uninformative individuals At the bottom of the pedigree. Anyone with no data and no offspring in the pedigree can be discarded. In the case of a nonfounder, the neighborhood indicator is turned off, and the individual is deleted from the linked list of children for the relevant marriage of each parent and the marriage is dissolved if the discarded individual is an only child. Founders on whom there are no data, and who become disconnected from the pedigree when their marriages are dissolved, are also dropped.

11 The McGibbs Programs and McLib Library

Information provided by Charles J. Geyer

11.1 Introduction

The original pedigree Gibbs sampler programs, for a discrete trait determined by a single diallelic locus, were written by Sheehan (1990). This code was incorporated into two directories by C. J. Geyer in 1991; the main program and its own subroutines were in a directory Gibbs, while the broader routines to set up Mendelian segregations, penetrances and to sample conditionally in the local pedigree neighborhoods were placed in a library directory Mendel (later called Mendel_Gibbs to avoid possible confusion with other program packages). Programs to implement Metropolis-coupled Gibbs samplers (Geyer, 1991) for these same pedigree Gibbs samplers were also implemented at this stage. In 1991-2, the need arose for more flexible versions of Metropolis coupling for problems relating to recessive traits on large complex pedigrees. It was necessary to use large numbers of coupled chains, and to use more complex penetrance functions for the models of these chains. In developing these more general samplers, the idea arose of coupling to an "independence chain" (GENE_DROP). It was hoped this would provide the sampler with the ability to sample a very complex space. Additionally, individual-specific priors were incorporated, and the normalising fac-

tors for the chains were computed. In conjunction with these developments, it was necessary to increase facilities for analysis of output and for diagnostics.

All these developments led to extensive revision of Gibbs and Mendel_Gibbs. In Fall 1992, these were replaced by McGibbs and McLib respectively; the main program directory and the associated library of subroutines implementing Metropolis-coupled Gibbs samplers for pedigree analysis. This revision was the work of C. J. Geyer.

11.2 The commands in McGibbs

The test commands in the current Makefile

make -i quick	quick (1 sec) test of test pedigree
make -i hops	quick (4 sec) test of hops pedigree
make -i hutt770	quick (80 sec) test of 770-member Hutterite pedigree
make -i hutt3315	quick (6.5 min) test of 3315-member Hutterite pedigree
make -i test	long (4 hr) test of test pedigree

The *.out file counts all the realizations in all the chains that are feasible under the correct model.

analyze test.bleat1 test.out1

1. calculates means and standard errors using batch means and the delta method.
2. gives warnings if the batches have "significant" lag 1 autocorrelation (they shouldn't). Not corrected for multiple testing, so the warnings may be ignored if there are many means and the autocovariance is only a little over 2 standard deviations above zero. See Geyer (1993) for more info.
3. for the test the exact answers (from peeling) are given in the file test.exact.

newtrait

1. makes trait files, works almost exactly like newtrait in PEDPACK

Compiling with -DGENE_DROP uses gene drop for the "hottest" chain. To test that change the top level Makefile adding -DGENE_DROP to both CFLAGS and LINTFLAGS and remake everything.

make -i short_drop	quick (1 sec) test of test pedigree
make -i long_drop	long (8 hr) test of test pedigree

The *.out file now counts all realizations, feasible or not, so only the "cold" chain has the correct distribution.

analyze long.bleat1 long.out1 1

1. calculates means and standard errors as before, except now only for the chain indicated by the third argument. 2. again the exact answers (from peeling) are in test.exact. The exact answers for all chains are in long.exact (though analyze doesn't calculate them).

11.3 Summary of some changes between Mendel_Gibbs and McLib

Everywhere (including mendel.h)	
1.	removed all references to flipptr, flippees, and ice_probs, INCLUDING from the argument list of full_estimate
2.	changed the definition of triplet to remove one indirect reference argument is now estimates rather than analysis_pointer-estimates
In copy_phenos.c	
1.	put "stderr" in fprintf – which corrects a BUG.
In full_compute.c	
1.	deleted child_ptr and marriage_ptr and j (not used).
2.	added variable "foo" – which corrects a BUG.
In full_estimate.c	
1.	deleted j (not used).
In full_mc_cycle.c	
1.	deleted j (not used).
2.	do Metropolis swap nchain / 2 per cycle.
3.	removed a bunch of commented out print statements.
In make_trips.c	
1.	deleted name (not used).
In random_mc_itr.c	
1.	deleted j, k, l, p, q, variance (not used).
2.	deleted reset_analysis(& results);
3.	deleted results.cycle_number;
In reset_anal.c	
1.	deleted i (not used).
In random_shuffle()	
1.	removed temp (not used).

Added feas_config.c
Moved random_shuffle() into random_infect.c
Moved random_draw() and full_compute() in to full_estimate.c

12 The Data files

12.1 Format for pedigree and phenotypic data

The pedigree input format of the data is standardized. The first row is contains four integers:

$$n \; w \; s \; t$$

n = number of individuals in each pedigree. If $w = 0$, there will be n rows of data, each consisting of s integers and t real numbers (float/double). If $w > 0$ this is a code that each line will contain additional integer indicators; this option is currently only used by the LINK_LIN programs.

The first three integers in each row consist of the data in triplet form; indicators of the individual, his father, and his mother. It is imperative (at this point) that the pedigree is ordered such that parents proceed offspring, because some routines assume this. Currently there is no check on the input to enforce this.

The fourth integer on each row normally indicates sex (where available) and the remainder can be used for fixed effects indicators (generation, age class, birth cohort etc.), or for discrete phenotypes. For linkage analysis programs such as *linkreg* the final column of integers is used for the marker phenotypic data; for missing marker phenotypes "999" is used.

The t real variables are t quantitative trait measurements on the individuals. Missing data values should have an integer part 999; this can be changed by redefining missing in the appropriate header files. The relevant header file is $(GIBBS)/Headers/quant.h for the programs in PolyEM and MIXED, where $(GIBBS) is simply the string /usr/users/pedpack/gibbs, or whatever it is reset to after installation. The relevant header file for Clip is head.h in $(GIBBS)/Clip. Currently, the Gibbs programs do not have this variable in the head.h file in $(GIBBS)/Gibbs; most of the Gibbs and McGibbs programs use 0 for missing values (this should be changed).

For the LINK_LIN programs additional information is necessary. The second integer w in the data file specifies the number of chains (see section 9.1). Each chain requires its own penetrance matrix. Further, to allow for individual-specific penetrances, each row of the data file will include w additional integers, specifying the penetrance matrix to be used for that individual in each chain. Note that the true (assumed) genetic model is given by the penetrance matrix 1. For the chain from which realizations under the true model are drawn, all individuals should have this penetrance matrix. For the other chains, performance may be enhanced if a modified penetrance is used only for some individuals. For example, the line

202 0 0 2 3 1 197.0 1 2

indicates that individual 202 is a founder (father = mother =0), female (sex=2), fixed effect level 3 (e.g. age-group), marker phenotype 1, and quantitative trait value 197.0. The final two integers show that $w = 2$, and that this individual 202 is to be accorded penetrance matrix 1 in the first chain, and penetrance matrix 2 in the second.

The additional lines appended to the end of the basic pedigree data input are the same as before (see section 8.4) except for the first additional line. There are three integers on this line, m, n, and q; where m is the number of joint trait and marker genotypes, n is the number of genotypes at the trait locus, and q is the number of alleles at trait locus.

For multiple markers, the pedigree data file is as described except that the first line of the data file has been changed to:

$$a \qquad b \qquad c[1] \qquad \cdots \qquad c[b]$$

where a = size of the pedigree; b =number of marker loci; $c[i]$=number of chains for the i^{th} marker locus. The individual entry lines are as before, except that the trait data is now followed by $c[1] + ... + c[b]$ integers, indicating the penetrance matrix to be used for the individual for each of the chains for each of the marker loci. Additionally, a penetrance matrix for the trait locus with d lines and e columns (allowing more general traits to be handled) is inserted between the model parameter specification and the output file names, where d = number of genotypes and e = number of phenotypes (see section 8.4).

12.2 Peeling sequences

Those programs which use the peeling routines for exact likelihood computation of either a polygenic or major-gene model require an additional input file. This file contains a *peeling sequence* which specifies the sequence of operations for an exact "peeling" computation of the likelihood on a pedigree without loops. Either a polygenic or a major gene (1 or 2 locus) model can be peeled. In the current implementation, one nuclear family is peeled in each operation, and each row of the data file corresponds to one operation. Since the pedigree has no loops, each *cutset* consists of a single individual who connects the family being peeled to the remainder of the as yet unpeeled pedigree; this individual in refered to as the *pivot*. The format for each row is:

parent-1 parent-2 pivot (# of children) (ID's of children)

The identifying numbers are the name ID's of individuals. The peeling programs assume that the two parents are in the order father, then mother. However this is normally irrelevent in peeling a genetic model, since peeling is done on residual observations after current estimates of fixed effects (including sex effects) are removed. Many of the older peeling sequences, including those generated automatically from the peel_seq program, give the parents in the order mother, then father. The pivot is the individual who connects to the remaining unpeeled portion of the pedigree, on whom a partial likelihood is held. The final pivot ID is given as zero, indicating to the program that peeling is complete and a final likelihood should be computed.

An example of a peeling sequence file may be found in $(GIBBS)/Data/Test/polyem_peelseq45. This short file contains the following:

31	25	25	4	42	43	44	45	
30	18	18	3	39	40	41		
14	29	14	4	35	36	37	38	
11	28	11	1	34				
11	27	11	1	33				
11	26	11	1	32				
8	21	8	1	25				
8	20	8	1	24				
8	19	8	1	23				
8	10	8	1	22				
9	5	5	3	11	12	13		
4	3	7	2	7	8			
7	6	6	5	14	15	16	17	18
2	1	0	2	5	6			

Thus the first peeling operation is to peel the nuclear family with parents with ID numbers 31 and 25; in this test file the parents are in the order mother then father. The 31 in the first row is the ID of the mother, and the 25 is the ID of the father. The father's ID is repeated in the third column in this case because he is the pivot. The fourth column corresponds to the number of children, and their ID numbers follow. Notice that the pivot ID in the final row is a zero, signaling that the peeling is complete.

Another test peeling sequence can be found in the file $(GIBBS)/Data/Test/mixed_peelseq. This is the peeling sequences for the 59-member pedigree used as a test example by the programs *mixd* and *linkreg* in MIXED. For this pedigree there are 15 sibships, and hence 15 peeling operations. The largest sibship peeled in a single operation has eight offspring.

Hannah Payne in 1992 wrote a C program that automatically generates peeling sequences on zeroloop pedigrees, from a list of pedigree triplets. This routine has recently been integrated into the Nghds library (see section 6.1) and a test program *test_peel_seq.c* has been added to MIXED to check working of the routine, but other main programs in the software do not, as yet, use the routine.

12.3 The marker data file

First, we specify the file for the case of a single linked markers with an arbitrary number of alleles. The first line of the additonal marker input file contains 4 integers, a, b, c, and d, where a=number of alleles at the marker locus, b=number of genotypes at the marker locus, c=number of phenotypes at the marker locus, and d=1 if allele frequencies are known and 0 if allele frequences are unknown. If d=1, then the next line will specify the known allele frequencies; $f_1, f_2, \cdots, f_a$. If d=0, this line is omitted. The following line contains an integer p, specifying the number of penetrance matrices to be used for analyzing the datam and then follow b p lines specifying the p penetrance matrices. Each line contains c probabilities $P(phenotype \mid genotype)$.

Here is an example of a marker input file in which the marker locus has three alleles, and hence six genotypes. There are also six marker phenotypes, and the marker allele frequencies are

known, and given. There are two penetrance matrices, the first (the identity matrix) is the "true" model. The second has modified non-zero penetrances for two heterozygous genotypes; analysis of the marker locus data shows this penetrance matrix will ensure irreducibility in this particular example.

```
3       6       6       1
0.239   0.760   0.001
2
1.0     0.0     0.0     0.0     0.0     0.0
0.0     1.0     0.0     0.0     0.0     0.0
0.0     0.0     1.0     0.0     0.0     0.0
0.0     0.0     0.0     1.0     0.0     0.0
0.0     0.0     0.0     0.0     1.0     0.0
0.0     0.0     0.0     0.0     0.0     1.0

1.0     0.0     0.0     0.0     0.0     0.0
0.02    0.9     0.02    0.02    0.02    0.02
0.0     0.0     1.0     0.0     0.0     0.0
0.0     0.0     0.0     1.0     0.0     0.0
0.02    0.02    0.02    0.02    0.9     0.02
0.0     0.0     0.0     0.0     0.0     1.0
```

For multiple marker loci, we have the following modifications. The first line specifies the number of alleles for the trait locus, which should be checked to make sure it is the same as the number specified in the input data file. The next three lines specify information about the markers. They are:

$$
\begin{array}{llll}
b & & & \\
m[1] & \cdots & m[b-1] & m[b] \\
r[1] & \cdots & r[b-1] &
\end{array}
$$

where b is, again, the number of markers, $m[i]$ = number of alleles for the i^{th} marker, and $r[i]$ = recombination fraction between a pair of consecutive markers i and $i+1$.

There follow are b blocks. Each block specifies information about one marker, as in the single-marker data file. Specifically, for each block, The first line contains 4 integers, g, hk, and l, where g=number of alleles at the marker locus; h=number of genotypes at the marker locus; k=number of phenotypes at the marker locus; l=1 if allele frequencies are know; 0 if allele frequencies are unknown. If l=1, then the next line will specify the known allele frequencies; $f_1, f_2, \cdots, f_g$. If $l = 0$, this line is omitted. The following line contains an integer p, specifying the number of penetrances needed for analyzing the current data set. It is then followed by h p lines specifying the p penetrances. Each line contains k probabilities $P(phenotype|genotype)$.

The following is an example of a marker input file for two marker loci, one with two alleles, and the other with three, at a recombination distance 0.22. The diallelic marker has codominant alleles, and only a single penetrance matrix. The triallelic marker is as above, and a second penetrance matrix is required to ensure irreducibility of the Markov chain samplers for this marker.

```
2

2
2       3
0.22

2       3       3       1
0.709   0.291
1
1.0     0.0     0.0
0.0     1.0     0.0
0.0     0.0     1.0

3       6       6       1
0.239   0.760   0.001
2
1.0     0.0     0.0     0.0     0.0     0.0
0.0     1.0     0.0     0.0     0.0     0.0
0.0     0.0     1.0     0.0     0.0     0.0
0.0     0.0     0.0     1.0     0.0     0.0
0.0     0.0     0.0     0.0     1.0     0.0
0.0     0.0     0.0     0.0     0.0     1.0

1.0     0.0     0.0     0.0     0.0     0.0
0.02    0.9     0.02    0.02    0.02    0.02
0.0     0.0     1.0     0.0     0.0     0.0
0.0     0.0     0.0     1.0     0.0     0.0
0.02    0.02    0.02    0.02    0.9     0.02
0.0     0.0     0.0     0.0     0.0     1.0
```

12.4 Multiple pedigrees and complex pedigrees

The programs were developed in the context of analyzing data on a single large pedigree, and, although development is continuing to expand their scope, most are still restricted to this case. For program testing purposes, multiple pedigrees can be input, and connected by unobserved offspring individuals; note that every individual in the data set must have a unique identifier. This addition of unobserved connecting offspring individuals will ensure that the programs perform correctly on the set of pedigrees, but the Monte Carlo analysis is very inefficient. Where data on disjoint components of a pedigree are independent, use should be made of this fact in the Monte Carlo estimation of expectations, both in MCEM and in MCLR. (If X and Y are known to be independent, one estimates $\mathrm{E}(XY) = \mathrm{E}(X)\mathrm{E}(Y)$ by the sample average of X-values times the sample average of Y-values, not by the sample average of products of X and Y!)

New programs are being developed to permit efficient analysis of pedigrees with multiple components. A new routine *components()* in the Nghds library can be called on input. This routine determines the disjoint component pedigrees, sets an *component* indicator in each individual, giv-

ing the component of which he/she is a member, and sets up linked lists of the individuals in each component. These structures enable sampling and likelihood estimation to be done on a component-by-component basis, but as yet this has only been implemented in the *autozyg* program in Autozyg.

Also in order here is a note about complex pedigrees and inbred individuals. While the Monte Carlo sampling performs correctly regardless of the complexity of the pedigrees, some caution should be observed. First, with inbreeding, the mixed model (especially with dominance variance in addition to additive genetic effects) is not well defined. Do not run the PolyEM programs with non-zero dominance variance, *sigd*, on pedigrees with inbreeding! The interpretation of any variance component model fitted on a pedigree with inbreeding is left to the user! Secondly some programs, primarily in PolyEM and MIXED used peeling computations as a tool in likelihood analysis, either to provide a more efficient Monte Carlo estimate, or to obtain exact likelihoods of simple major-gene or polygenic models. The programs currently implemented in the Peelie section of the Quant library permit polygenic or major gene peeling, but for zero-loop pedigrees only. With a looped pedigree it is necessary to use versions of the programs which do not require peeling. We have recently implemented a major-gene joint peeling program, in C, (translating from the Fortran programs of Thompson, 1977), but this is not yet integrated into the current program libraries.

12.5 Test Data

Test data is found in the directory $(GIBBS)/Data/Test. The purpose of having the test data is to check successful installation of the programs by testing their output against a "gold standard". It is recommended that the programs be run first on the test datasets verify that you are obtaining the same results. It can also be useful to look at the format of the test data to check your comprehension of the instructions about formatting given in the Format section. The test data directory should be kept in place so that new versions of the programs can be tested against a standard after they have been installed.

12.6 Creating Data Directories

The appropriate place to put new data is in one or more subdirectories of $(GIBBS)/Data. For example, make a subdirectory of $(GIBBS)/Data called NewData, and put datasets in this directory. The Makefiles for each program will have to be edited so that the programs read the data from this new directory. Of course, each individual user may have their own data in such a directory, and/or the manager of the package may keep a centralized set of data subdirectories (or both). Users must modify Makefiles so that the correct data sets are used in a given application. This is explained in detail in the Makefiles section.

On the version of this package installed at University of Washington, additional data directories include *Hutt*, *GAW*8 and *TMO*. The data sets in these directories are NOT available for distribution, even though the distributed example Makefiles may contain references to them.

13 The programs on a VMS machine

13.1 Putting the programs on a VMS machine

The following instructions are intended to help with setting up the programs on a computer running under the VMS operating system. The information in this section was provided by Ellen Wijsman.

Distinctions between VMS and UNIX which have been found to be important in getting the programs running are the following. For purposes of comparison with UNIX, it should be noted that VMS is not case-sensitive, so that the command:

dir random.seed

is identical to the command:

dir Random.seed

It is also useful to note that VMS maintains version numbers of files, so that files like random.seed will need to be periodically purged. Also, paths and directories are indicated with brackets and dots rather than slashes; e.g.,

dir [root.new_gibb.mixed]

will list all files in the mixed subdirectory under the new_gibb subdirectory in the account of root. The equivalent under UNIX would be:

ls root/new_gibb/mixed

Finally, the default size of the exponents under VMS is 38. To get larger exponents (308), compile with the g_float option:

cc/g_float subroutine.

It is important to note that in general computations should not exceed the default limits, so that over- or underflow with the default limit but not with the expanded limit can be diagnostic of a problem.

In order to get the programs running, the following will work, although it is not the only way in which the programs can be set up and tested. More elegant solutions are possible, and it is possible to find programs which will "translate" makefiles for UNIX into something which works more or less the same under VMS.

(1) Set up an appropriate directory tree, so that directories under VMS will correspond to those in the makefiles for UNIX. For purposes of discussion, we will assume an account (user) name of

"root", and that the basic directory for the gibbs programs is called "new_gibb". At the time of this writing, the directory tree included:

[root.new_gibb.data]

[root.new_gibb.headers]

[root.new_gibb.libs]

[root.new_gibb.mixed]

[root.new_gibb.nghds]

[root.new_gibb.gibbs]

as well as a number of other directories which parallel those indicated in the UNIX makefile. The test data files are in [root.new_gibb.data]; the .h files are in [root.new_gibb.headers], libraries common to multiple programs are in [root.new_gibb.libs], and various source code files are in the appropriate directories labelled, e.g., [root.new_gibb.mixed].

(2) In each of the directories in which there are source files, set up a command file, e.g., compile.com, which compiles each subroutine in the directory. For example, in the mixed directory, compile.com would be:

```
$ set def [root.new_gibb.mixed]
$ cc smpl_ga/include_dir=[root.new_gibb.headers]
$ cc mixed/include_dir=[root.new_gibb.headers]
```

etc. for each source file

(3) After all subroutines have been compiled, put all both those containing main subroutines into libraries - one library per directory, and store the libraries which are needed by multiple programs in [root.new_gibb.libs]. Keep the program-specific libraries with the source code. Then the programs in each program directory can be linked, for example:

```
link mixd,mixed/lib,[-.libs]quant/lib,[-.libs]nghds/lib,[-.libs]forsubs/lib,[-.libs]rans/lib
```

(4) To run a program, you need to use the following sequence of commands (e.g., for the mixd program), which again can be put in a command file, e.g., run_mixd.com:

```
$ set def [root.new_gibb.mixed]
$ mixd == "$usr:[root.new_gibb.mixed]mixd.exe"
$ mixd [root.new_gibb.data.test]mixed_mixd. [root.new_gibb.data.test]mixed_peelseq.
```

In this example, note the "." in the file names, which VMS must have but which is optional under UNIX. Also note thate the part of the path "$usr:" is necessary, and is the disk name (logically defined) on which the program "mixd.exe" resides.

While the directory tree and details for how the programs are compiled and linked is subject to any amount of reorganization, this way of running the programs seems to be the only way under VMS of getting the data files included in the command line for the program.

13.2 CMLIB routines needed for running the programs

Some of the programs can be run without access to **cmlib**, but others, particularly the PolyEM programs, use **cmlib** routines for matrix manipulations and computations. The simplest solution is to acquire **cmlib** which is publicly available. However, for those who do not wish to install a full **cmlib** the following is a list of the Fortran routines actually used. Fortran source code is publicly available.

BALANC, BALBAK, DASUM, DAXPY, DDOT, DGECO_, DGEDI_, DGEFA, DSCAL, DSWAP, ELMHES, ELTRAN, HQR, HQR2, IDAMAX, R1MACH, RG_, RS_, TQL2, TQLRAT, TRED1 and TRED2

References

Cannings, C., Thompson, E. A., and Skolnick, M. H. (1978) Probability Functions on Complex Pedigrees. *Adv. Appl. Prob.* 10:26–61.

Dempster, A. P.; Laird, N. M.; Rubin, D. B. (1977) Maximum likelihood from incomplete data via the EM algorithm, *J R Stat Soc. B* 39:1–38.

Elston, R. C. and Stewart, J. (1971) A General Model for the Genetic Analysis of Pedigree Data. *Human Heredity* 21:523–542.

Geyer C. J. (1991a) Markov chain Monte Carlo maximum likelihood. *Computer Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, Pp 156-163. Interface Foundation of North America.

Geyer CJ. (1991b) Reweighting Monte Carlo Mixtures. Technical Report No. 568, School of Statistics, University of Minnesota.

Guo, S. W. (1991) Monte Carlo methods in the quantitative genetics. Unpublished Ph.D. dissertation. Dept. of Biostatistics, University of Washington.

Guo, S. W. and Thompson, E. A. (1991) Monte Carlo estimation of variance component models. *IMA J Math Appl in Med & Biol* 8:171-189

Guo, S. W. and Thompson, E. A. (1994) Monte Carlo estimation of mixed models for large complex pedigrees. In press; Biometrics.

Guo, S. W. and Thompson, E. A. (1992). Two papers on the Monte Carlo estimation of models for complex genetic traits. Technical report No. 229, Department of Statistics, University of Washington.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57: 97–109.

Hasstedt, S. J. (1982) A mixed-model likelihood approximation on large pedigrees. *Computers and Biomedical Research* 15:295–307.

Hasstedt, S.J. and P. Cartwright. (1979) *PAP – Pedigree Analysis Package.* Technical report No. 13, Department of Medical Biophysics and Computing, University of Utah.

Henderson, C. R. (1976) A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. *Biometrics* 32:69–83

Knuth, D. E. (1981) *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 2nd ed.* Reading, Mass.: Addison-Wesley.

Lander, E.S. and Botstein, D. (1987). Homozygosity mapping: a way to map human recessive traits with the DNA of inbred children. *Science* 236:1567–1570.

Leppert, M. F., Hasstedt, S. et. al. (1986) A DNA probe for the LDL receptor gene is tightly linked to hypercholesterolemia in a pedigree with early coronary disease. *American Journal of Human Genetics* 39:300–306

Lin, S. (1993) *Markov chain Monte Carlo estimates of probabilities on complex structures.* Ph.D. Thesis, University of Washington.

Lin, S., Thompson, E. A., and Wijsman, E.M. (1993) Achieving irreducibility of the Markov chain Monte Carlo method applied to pedigree data. *I.M.A. J. Math. Appl. Med. & Biol.,* 10:1–17.

Lin, S., Thompson, E. A., and Wijsman, E.M. (1994) Finding non-communicating sets for Markov chain Monte Carlo estimations on pedigrees. *American Journal of Human Genetics* In press.

Marsaglia, Ananthanarayanan and Paul (1973). 'Super-Duper' random number generator package. School of Computer Science, McGill University, Montreal, Quebec.

Oram A. and Talbott S. (1993). *Managing Projects with Make,* O'Reilly and Assoc. Inc., Sebastopol CA.

Sheehan, N. A. (1990) Genetic restoration on complex pedigrees. PhD dissertation. Dept. of Statistics, University of Washington.

Sheehan, N. A. and Thomas, A. W. (1992) On the irreducibility of a Markov chain defined on a space of genotype configurations by a sampling scheme. *Biometrics:* in press.

Smith, C.A.B. (1953). Detection of linkage in human genetics. *Journal of the Royal Statistical Society (B)* 15:153–192.

Thompson, E. A. (1977) Peeling programs for pedigrees of arbitrary complexity. Tech Report No. 6, Dept Biophysics, University of Utah.

Thompson, E.A. (1986) *Pedigree Analysis in Human Genetics*. The Johns Hopkins University Press, Baltimore.

Thompson, E.A. (1988) Two-locus and three-locus gene identity by descent in pedigrees. *IMA Journal of Mathematics Applied in Medicine and Biology* 5:261–280.

Thompson, E. A. (1991) Probabilities on complex pedigrees; the Gibbs sampler approach. *Computer Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, Pp 321-328. Interface Foundation of North America.

Thompson, E. A. and Guo, S. W. (1991) Evaluation of likelihood ratios for complex genetic models. *IMA J. Math. Appl. Med. Biol.* 8:149-169.

Thompson E. A., Shaw R. G. (1990) Pedigree analysis for quantitative traits: variance components without matrix inversion. *Biometrics* 46:399-413

Thompson, E.A. and Shaw, R.G. (1992) Estimating polygenic models for multivariate data on large pedigrees. *Genetics* 131:971-978.

Thompson, E. A. and Wijsman E. M. (1990). The Gibbs sampler on extended pedigrees: Monte Carlo methods for the genetic analysis of complex traits. Technical report No. 193, Department of Statistics, University of Washington.

Thompson EA, Lin S, Olshen AB, Wijsman EM. (1993). Monte Carlo analysis on a large pedigree. In *Genetic Analysis Workshop 8. Genetic Epidemiology,* to appear.

Thompson, E. A. (1994a). Monte Carlo likelihood in genetic analysis. In: *Probability, Statistics, and Optimization*, F. P. Kelly, ed. In Press.

Thompson, E. A. (1994b). Monte Carlo likelihood in the genetic analysis of complex traits. *Transactions of the Royal Society, B;* In Press.

Thompson, E. A. (1994c). Monte Carlo likelihood in genetic linkage analysis. *Statistical Science;* Submitted.