29th November, 1995.

GAS Manual

General User Guide

2.0

Chapter 1 : Introduction

This is the general manual for the Genetic Analysis System version 2.0. The analysis modules are described in a companion volume. To supplement this manual there are a series of demonstration examples available in the form of computer files, and it is highly recommended that these are used in conjunction with the text.

1.1 Contents

1.2 Obtaining GAS

For information on the latest version of the *GAS* program, manuals, and example files connect via WWW to

`http://info.ac.ox.uk/~ayoung/gas.html`

GAS is currently available on the following systems: IBM-PC DOS, Vax VMS, DEC Ultrix, IBM AIX, DEC Alpha OSF/VMS, SUN Solaris, SGi IRIX and SunOS.

1.3 Legal Notice

The *GAS* program, manuals and associated files are not public domain and may not be modified without express permission of the author. However they may be used and copied freely provided that:

(1) this notice is distributed along with the program,

(2) no charge is made to other parties for services reliant upon *GAS* without express permission from the author,

(3) you agree that neither the author, nor any institution with which he has been associated, accepts responsibility for either the accuracy of results produced by *GAS* nor for the use to which they are put,

(4) the use of *GAS* is acknowledged in any publications incorporating results produced using the program.

With respect to clause (4), the following form of words is suggested:

"data was analyzed using the GAS package version 2.0

(c) Alan Young, Oxford University, 1993-95."

You should also include any relevant references given in the *GAS* manuals.

1.4 Style

Examples of input files are shown `in square font`. File-names are shown *in slanted roman font*. General parameters for functions are shown *in italics*. Note that *GAS* converts all input to lower-case unless it is enclosed in "quotation" marks.

1.5 Errors

During operation the *GAS* program may produce four types of message:

INFO	information, note appended to output file
WARNING	possible error, note appended to output file
ERROR	serious error, exit from current *GAS* module
FAILURE	fatal error, immediate exit from program

after *GAS* has completed, a copy of these messages can be found in the logfile (assuming you have requested one with the `set logfile` command).

If the program terminates with the message "INTERNAL ERROR" then an internal trap has been triggered and the program will need to be modified and re-compiled to run on your system. In this case please notify Alan Young via email at "ayoung@vax.ox.ac.uk" and include the program version, the computer system, and the actual input files that were being used when the failure occurred.

1.6 Examples

To supplement the manuals there are a series of demonstration examples included as computer files in the distribution package, and it is highly recommended that these are used in conjunction with the text. Output files ending in '.ps' contain Postscript graphics and must be viewed either on a screen previewer or by sending them to a Postscript compatible printer. If you're learning how to use *GAS* then a good method is to modify the example files to explore how the various optional parameters affect the type of analysis performed.

1.7 Help!

This manual covers operation of *GAS* only, and is not intended to form a statistics text. References have been included for most of the analytical routines, and these should be consulted for further information on their applicability. Please remember: *GAS* is designed to perform fast accurate analyses of genetic data, but the results will only be useful if an appropriate choice of routines has been made.

The file *gas.faq* contains answers to some of the commoner questions that users have asked about *GAS*. If you have a query, it is suggested you first post it to the bulletin board bionet.molbio.gene-linkage, and if no suitable response arrives then email:

ayoung@vax.ox.ac.uk

Chapter 2 : Running the Program

The *GAS* program is controlled by giving it a list of commands contained in a file called a 'gasfile' (this is analogous to a 'BAT' file on PCs, or a 'COM' file on VAX systems).[1] These commands describe where to look for data and what to do with it.

2.1 Order of Input

The program requires that the following data be entered:

- [] Locus Specifications
- [] Control Variables
- [] Pedigree Specifications
- [] Modifications required
- [] Analysis Routines to be used

Comments may be placed in any input files by preceding them with an exclamation mark (!), which causes the program to ignore the remainder of the current input line. The letter `x` is usually used to mark where data is unknown. Individual program commands may be written over several lines, and should end with a semi-colon ;

2.1.1 Locus Specification

This is the specification and labelling of the loci types and other variables which have been measured for individuals in the pedigree. This is described in Chapter 4.

2.1.2 Control Variables

These variables modify the performance and output of the program and are altered using the `set` command. They are described in Chapters 6 and 8, with a summary given in Appendix C.

2.1.3 Pedigree Specification

For each subject there must be an entry indicating to which family they belong, their parents, sex, and what is known about their genotypes and phenotypes. This is described in Chapter 5.

[1] This method was chosen to ensure that *GAS* will perform identically on a variety of computer systems. Note that a few results may differ slightly between systems due to different numeric 'rounding' conventions used by their output routines.

2.1.4 Modifications Required

GAS can modify the contents of a dataset to make it suitable for different purposes. The methods for doing this are described in Chapters 9, 10 and 11.

2.1.5 Analysis Routines

This is a list of commands telling the *GAS* program what sort of analysis to perform on the data. These are described in a companion document to this manual.

2.2 On-line Help

GAS has a basic help facility built into the program. Typing **`gas help`** will give a list of the main commands used by the program. When running the program, a reminder of the parameters for most of the commands and routines can be obtained by replacing their normal argument with **`help`**. For instance **`set help;`** and **`call sibdes( help );`** will respond by listing the parameters that may be used with the **`set`** and **`sibdes`** commands respectively.

Example 2.1

Below is a basic gasfile which loads in a dataset and runs a sib-pair analysis on it. The file is called *basic.gas* and you can run it by typing '**gas basic**'.

```
set logfile = basic.log;                    ! 1
set outfile = basic.out;                    ! 2
read( data basic.loc );                     ! 3
read( pedigree basic.ped );                 ! 4
program;                                    ! 5
call sibdes( locus atopy mk1 mk2 );         ! 6
stop;                                       ! 7
```

The lines in this control file perform the following actions:

(1) a logfile is opened. A copy of any messages generated by *GAS* will now be copied to the file *basic.log*.

(2) an outfile is opened. The output from the following sib-pair analysis will be sent to the file *basic.out*.

(3) data describing the loci involved is read from the file *basic.loc*.

(4) data on the pedigree structure is loaded from the file *basic.ped*.

(5) the **program** command tells *GAS* to expect a list of analytical operations to be performed.

(6) the **sibdes** routine is called to perform an IBD sib-pair analysis on the loci atopy, mk1 and mk2 (which are described in the data files).

(7) the **stop** command marks the end of instructions to *GAS*.

After running this example look at the files it used (*basic.loc* and *basic.ped*) and created (*basic.log* and *basic.out*). As with all the examples in the manual, feel free to modify any of the files to see how the output changes — if you enter anything that doesn't make sense to the *GAS* program, it will tell you where to look in the input files for mistakes (the filename and line-number will appear in the logfile).

Chapter 3 : Data Input and Output

The first stage in any analysis is to load data into the program using the **read** command.

3.1 Reading Data

GAS is able to read locus and pedigree data in two main formats. The preferred format is a keyword based system, called 'g-format'. An alternative is that used in the LINKAGE suite of programs denoted by 'l-format'.[2] Some of the exercises later in this manual show how to convert data between these. To load data into *GAS* use the **read** command thus:

read(*file_type file_name(s)* **);**

where *file_type* describes the type of data being loaded (the options are listed below) and *file_name(s)* is a list of one or more files from which data is to be taken.

data	locus specifications in g-format
ldata	locus specifications in l-format
pedigree	family data in g-format
lpedigree	family data in l-format
alsize	family data using CA-repeats

Hence the command

```
read( pedigree bigfam.ped );
```

will read family data in g-format from the file *bigfam.ped*. If you wish to read and/or write in a directory other than the current one, then you must enclose the path and file name in quotation marks, thus

```
read( pedigree "../raw/bigfam.ped" );
```

would (under unix) direct *GAS* to go up one directory level then look in the sub-directory *raw* for the file *bigfam.ped*.

The **pedigree** and **lpedigree** formats are discussed in Chapter 5, and the **alsize** format[3] is discussed in Chapter 7.

[2] For a description of l-format see a manual for the LINKAGE programs.

[3] Generally used for reading CA-repeat marker data from automated fluorescent genotyping machines.

3.2 Writing Data

The `write` command may be used to send locus and pedigree held inside *GAS* into files for long term storage or processing by other programs. The basic format is

`write(` *file_type file_name* `);`

where *file_type* is one of:

`data`	`locus specifications in g-format`
`ldata`	`locus specifications in l-format`
`pedigree`	`family data in g-format`
`lpedigree`	`family data in l-format`
`program`	`analysis commands`

and *file_name* is the name of the file into which the data is put. For the first four file types, further control of the output is available using the `locus` parameter thus

`write(` *filetype file_name* `locus` *locus_name(s)* `);`

which causes only those loci whose names are listed to appear in the new file (subsets of data may also be written using the `edit`, `select` and `delete` features described later).

Example 3.1

The gasfile *io.gas* reads l-format data from the files *io.dat* and *io.fam*. It writes all of it in g-format to the files *io1loc.new* and *io1ped.new*, then only those loci corresponding to phenotypic data are written to the files *io2loc.new* and *io2ped.new*.

3.3 Nested Files

Because an input file of type **data** is simply a list of *GAS* instructions,[4] it may be used to load further files (and these files may also load further files, etc.).

A convenient way to organise the input files is to keep the genotype specification, the pedigree data and the program commands in three separate files (eg. *locs.gas*, *fam.gas* and *prog.gas* respectively), each of which may load further files. The **read** command can then be used in the main file to input the contents of each of these separate files.

```
read( data locs.gas );
read( pedigree fam.gas );
read( data prog.gas );
```

Using this hierarchical organisation, only the *prog.gas* file would need to be altered to perform different operations on the data. Similarly, any pre-written commands could also be stored separately. Schematically this organisation looks like

$$
main.gas \begin{cases} locs.gas & \qquad\qquad\qquad \textit{genotype specifications} \\[2ex] fam.gas & \begin{cases} fam1.ped & \textit{1st family} \\ \quad\vdots & \\ famN.ped & \textit{Nth family} \end{cases} \\[2ex] prog.gas & \begin{cases} alpmain.gas & \textit{main program commands} \\ alpsub.gas & \textit{subroutines} \end{cases} \end{cases}
$$

Alternatively, since the 'top' three files are all gasfiles, the file *main.gas* could be omitted and the same effect achieved by entering

```
gas locs fam prog
```

on the command line.

[4] The **include** command may also be used for this. The syntax is **include** *filename***;**.

Chapter 4 : Describing Loci

Before any data on individual subjects can be loaded into *GAS* it is necessary to specify what the data will consist of — ie. the names and types of variables which have been measured. Usually this will consist of a list of marker loci together with one or more hypothesized phenotype-influencing loci thought to be relevant to the genetic situation.

The specifications for each of the loci types (affection, binary, quantitative and named) follow a common pattern:[5]

> `set locus` *locus_name type_of_locus*
> *number_of_alleles gene_frequencies...*
> *specifications...;*

The `newlocus` command can be used to construct additional loci with values calculated according to various criteria — see Chapter 11 for more details on this.

4.1 General Parameters

4.1.1 Sexlinked

If the `sexlinked` qualifier is given after a particular locus, then the data for that locus is assumed to be sex-linked, and it may be necessary to enter additional parameters to describe this behaviour.

4.1.2 Eqfreq

The `eqfreq` qualifier may be used when the number of alleles is known, but their frequencies are irrelevant. For instance

```
8 eqfreq
```

is equivalent to entering

```
8 0.125 0.125 0.125 0.125 0.125 0.125 0.125 0.125
```

[5] Note that in the input file, although the parameters must appear in the same order as listed here, they can be split differently across lines of the file

4.2 Affection

Affection loci are used to describe a simple phenotype in which a subject is either affected or non-affected by some condition. Subjects may be divided into a series of liability classes corresponding to distinct groups within a population, within which the penetrance of the phenotype associated with the locus is assumed to act differently. The format required to specify an affection locus is:

set locus *locus_name* **affection**
number_of_alleles *gene_frequencies...*
number_of_liability_classes
name_of_liability_class_1 *penetrances...*
⋮
name_of_liability_class_N *penetrances...*;

If the locus has only one liability class then it's name is omitted from the specification. Often[6] the liability classes and penetrances are irrelevant to the type of analysis being performed, in which they may be replaced with the word `noclass` (equivalent to specifying one class with uniform penetrances of 0.5) thus

set locus *locus_name* **affection**
number_of_alleles *gene_frequencies...*
noclass;

[6] Currently only the `dissect` and '`lik`...' modules use the penetrance data.

4.2.1 Affection Loci Penetrances

Suppose a locus has n alleles and label the penetrance corresponding to a subject having i and j alleles as $p_{i,j}$. Then the penetrances can either be entered in triangular order as

$$name\_of\_liability\_class \quad p_{1,1}p_{1,2}\ldots p_{1,n}p2,1\ldots p_{2,n-1}p_{3,1}\ldots p_{3,n-2}p_{4,1}\ldots\ldots p_{n,n}$$

or alternatively in full square format in the order

$$name\_of\_liability\_class \quad p_{1,1}\ldots p_{1,n}p2,1\ldots p_{2,n}\ldots p_{n,1}\ldots p_{n,n}$$

the second format may be used to describe an imprinted locus if $p_{ij} \neq pji$ since in this case p_{fm} is taken to be the penetrance when a subject inherits the f allele from it's father and the m allele from it's mother.

For instance, to describe an affection locus, named "a1", with 2 alleles (or frequencies 0.75 and 0.25) and 3 non-imprinted liability classes (labelled `1`, `2` and `3`), enter the following:

```
set locus      a1 affection
2 0.75 0.25
3
1              0.3  0.79  0.90
2              0.3  0.90  0.95
3              0.5  0.13  0.37;
```

If this locus had only the first liability class, then it's specification would be:

```
set locus      a1 affection
2 0.75 0.25
1
               0.3  0.79  0.90
```

If the subject's trait value or liability class is unknown, all the penetrances are taken to be 1.

4.2.2 Affection Loci and Sex-linkage

For sex-linked data a further series of values must be entered for each liability class to describe the penetrances in male subjects (the first data is assumed to refer to females with 2 chromosomes).

For each liability class, a line is required of the form

$$name\_of\_liability\_class \quad p_1 p_2 \ldots p_n$$

where p_i is the penetrance of the condition in a male subject having the single allele i. Thus for a non-imprinted sex-linked locus having 3 alleles and 2 liability classes, called **child** and **adult** enter the following:

```
set locus      a1 affection
3 0.35 0.4 0.25
2
child          0.3  0.79  0.90 0.35  0.64  0.90
adult          0.3  0.90  0.95 0.20  0.50  0.64
sexlinked
child          0.3  0.65  0.00
adult          0.3  0.70  0.15;
```

again, omit the name of the liability class if there is only one. If the `noclass` qualifier is used, then `sexlinked` should be entered immediately after it.

4.3 Binary

Binary loci are used to describe a phenotype in terms of the presence or absence of several criteria (called factors) simultaneously. The format required to specify a binary locus is:

set locus *locus_name* **binary**
number_of_alleles *gene_frequencies...*
number_of_factors
code for allele_1...
⋮
code for allele_N...;

where `N` is the number of alleles. For instance, to describe a binary locus, named "bn1", with 4 alleles and 3 factors, enter the following:

```
set locus      bn1 binary
4 0.35 0.25 0.1 0.3
3
1  0  0
0  1  0
0  0  1
0  1  1;
```

Thus the line `1 0 0` specifies that subjects containing a copy of allele 1 will be positive for the first factor but not for the second and third factors (though they may have these latter two due to their other allele).

Hence a subject who is positive for the first and third factors and definitely negative for the second factor must have genotype '`1 3`'. However if their status for the second factor was unknown then the subject might have either genotype '`1 3`' or '`1 4`'.

4.4 Quantitative

Quantitative loci are used to store phenotypic variables which cannot be classified as simply present or absent (for instance, the height of a subject). The format required to specify a quantitative locus is:

set locus *locus_name* **quantitative**
number_of_alleles *gene_frequencies...*
number_of_liability_classes
name_of_liability_class_1 *penetrance_distributions...*
⋮
name_of_liability_class_N *penetrance_distributions...*;

The parameters from *number_of_liability_classes* onwards are only used by the `lik...` routines, and if these are not being used they can be replaced by the word `noclass` when creating a new locus. For instance, to describe a quantitative locus, named 'qval', with 2 alleles and no liability class information enter the following:

```
set locus     qval quantitative
2 0.65 0.35
noclass;
```

4.4.1 Quantitative Loci Penetrance Distributions

GAS has two forms for penetrance distributions[7] based around `constant` and `normal`[8] distributions. To specify that the penetrance associated with a genotype is independent of the quantitative value, use the syntax

`constant` *value*

and to specify that a penetrance has a normal distribution (based on the subject's measured trait value) use the syntax

`normal` *mean variance*

The penetrances associated with each genotype are entered in the order described above for affection loci penetrances. *GAS* supports multiple liability classes for quantitative traits,[9] and to describe a quantitative locus, called 'qval1', with 3 alleles and only one liability class (in which case the class name is omitted) you might enter the following:

[7] If you don't know what these are, then get statistical advice before using them!
[8] Also known as the 'Gaussian Distribution' or 'Error Function'.
[9] The LINKAGE programs support only one class.

```
set locus       qval1 quantitative
3 0.25 0.4 0.35
1
normal          22.3 5
normal          13.5 2.6
normal          23.1 1.64
constant        0.3
normal          -7.3 14.5
constant        0.5
```

which shows that the penetrance associated with the genotype '`1 2`' has a normal distribution with mean 13.5 and variance 2.6, and the penetrance associated with genotype '`3 3`' is fixed at 0.5

Similarly, to describe a quantitative locus, named 'qval2', with 2 alleles and 2 liability classes (labelled 'black' and 'blue') you might enter the following:

```
set locus       qval2 quantitative
2 0.25 0.75
2
black           constant        0.2
                constant        0.2
                constant        0.95
blue            normal          -25 12.4
                normal           0.0 14
                normal           24 12.4
```

If the subject's trait value or liability class is unknown, all the penetrances are taken to be 1.

4.5 Named

A named locus is one in which the alleles are definitely known and have individual names (or numbers). This classification is used to describe such things as markers, CA-repeats and l-format numbered loci. The specification for a named locus is:

set locus *locus_name* **named**
number_of_alleles *gene_frequencies...*
optional_data...
⋮ ;

For instance, the basic way to describe a locus `n1` with 6 alleles (individually identified as 1, 2, 3, 4, 5, and 6) is

```
set locus      n1 named
6 0.1 0.15 0.05 0.4 0.22 0.08;
```

The *optional_data* may be the names of the individual alleles (`name`), their size (`size`) (used when alleles are distinguished by their length in terms of DNA bases), variation around mean size (`range`), minimum size (`minsize`), their maximum size (`maxsize`), or the `sexlinked` parameter.

4.5.1 Allele Names

By default it is assumed that the alleles are labelled from 1 to the maximum number. However the `name` parameter may be used to give the alleles other labels, hence to describe a locus (called `gre`) in which the alleles are labelled as the first six greek letters, enter:

```
set locus      gre named
6 0.1 0.15 0.05 0.4 0.22 0.08
name alpha beta gamma delta zeta eta;
```

4.5.2 Named Loci and Sex-linkage

If the `sexlinked` parameter is given then the `read( pedigree`... `)` command will only expect one allele name for each male subject rather than the normal two.

4.5.3 Nodata

The **nodata** qualifier may be given after the frequencies. In this case no input data for this locus is expected and all subjects have the value set as unknown x.

This qualifier is useful before **read(alsize**...) to set up a blank locus in every subject which will be filled as new values are read.

4.5.4 Nofreq

The **nofreq** qualifier may be used with **nodata** when the frequency and number of alleles are unknown (generally when loading allele-size data with the **read(alsize);** option. It is equivalent to specifying two alleles of equal frequency.

4.5.5 Size and Range

The alleles of some named loci may be distinguished by their physical length along a chromosome, and the *GAS* program can read data in terms of such measurements using **read(alsize);**. If the expected lengths of the alleles are known prior to loading pedigree data, then they may be entered using either the **size**/**range** or **minsize**/**maxsize** options.

The **size** qualifier is used to describe the average length of each of the named alleles, and the **range** qualifier describes the range about each mean value which is likely to be measured. Hence

```
set locus      ns named
3 0.3 0.2 0.5
size 120.7 122.5 130.0
range 0.5 0.5 0.8
nodata;
```

describes a named locus **ns** with three alleles, the sizes of which are 120.7 ± 0.5, 122.5 ± 0.5 and 130 ± 0.8 respectively. The **nodata** parameter means that no data will be expected in files read using **read(pedigree**...) or **read(lpedigree**...).

4.5.6 Minsize and Maxsize

The `minsize` and `maxsize` qualifiers are used to describe the minimum and maximum sizes of each of the alleles present. Hence

```
set locus      nm named
2 0.3 0.7
name bigun littlun
minsize 140.7 122.5
maxsize 141.8 123.9;
```

describes a locus with 2 alleles, called `bigun` and `littlun` which have lengths in the ranges 140.7 to 141.8 and 122.5 to 123.9 respectively.

Example 4.1

The file *xloc.gas* contains descriptions of each type of locus in g-format, and the file *xloc.ped* shows an example family corresponding to them. Run *GAS* to write out the locus data in in l-format to *xlocl.new* and *xlocped.new*. Note the use of `edit( lformat )` to convert alphabetic labels to numbers, which stores a table of correspondences in the logfile. Warnings will be given for variables in which the information content has to be reduced in order to convert them to l-format.

Example 4.2

The file *xlocl.gas* reads in the l-format data produced in the previous example, and writes it to the g-format files *xlocg.new* and *xlocgped.new*. Compare these files with the input files in the previous example.

Chapter 5 : Describing Subjects

After the type of input data has been described, either by entering it directly into the *GAS* control file or else by loading it in using **read(data**...) or **read(ldata**...), the information for the actual subjects must be loaded. This is done by placing it in one or more files and reading them using either **read(pedigree**...) or **read(lpedigree**...)[10] (to load data in terms of allele sizes, see Chapters 3 and 7). The general syntax used to describe a subject is:

ped_name name parent_1 parent_2 sex loci... optional_modifiers...

Note that all of the data for a single subject must appear on the same line of the input file, and that the loci must be listed in the same order they were specified earlier.[11]

5.1 Relationship

The first four entries on each line correspond to:

(1) the family of which the subject is a member,

(2) the name of the subject,

(3,4) the names of the subject's parents.

These names can any combination (up to 16 characters long) of the letters a-z, numbers 0-9 and the underscore (_) character, however if they begin with a number then they must be wholly numeric.[12] If a parent is unknown (ie. not listed elsewhere in the pedigree) then it's name should be replaced with `x`.

5.2 Sex

The 5th column describes the sex of the subject and is either `m` for males, or `f` for females.

5.3 Loci

For each of the loci described in the genotype specification stage (see Chapter 4) there must be a corresponding entry for each subject (unless the **nodata** qualifier was used, in which case the entry for that locus must be omitted).

[10] l- and g-format data can be mixed, but not in the same file.
[11] To get round this latter restriction see the `locus` parameter in Chapter 7.
[12] For instance `314`, `fred`, `big_bad_alice` and `John6` are all allowable, but `1b` is not.

5.3.1 Affection Loci

The format for an affection locus with more than one liability class is:

status liability_class

where *status* is one of `y`, `n` or `x`. If there is only one liability class this is reduced to

status

For example, suppose locus '`baldness`' has classes '`blonde`', '`auburn`' and '`brown`', then to say that a subject has brown hair and is affected, the following should be entered:

```
y brown
```

For an non-affected person in the same liability class (ie. `brown`) the entry is:

```
n brown
```

If the status is unknown, the subject is described as:

```
x brown
```

5.3.2 Binary Loci

A binary locus is given in terms of the results applied (ie. the factors), where these have the three possible results positive (`y` or `1`), negative (`n` or `0`) and unknown/not-tested (`x`). The format for a binary subject locus with `N` factors is thus

result_1 result_2 ... result_N

Hence for a locus called '`test_result`' in which factors 1 and 5 are definitely negative, factors 2 and 3 are definitely positive and factor 4 is unknown (either because it was not tested, or the results were ambiguous) enter the following:

```
n y y x n
```

or alternatively

```
0 1 1 x 0
```

5.3.3 Quantitative Loci

A quantitative locus is specified in terms of the quantity it measures (and the liability class if more than one) hence the format is:

quantity liability_class

Thus for a locus having value 12.5 for a particular subject and only one liability class the entry would be:

```
12.5
```

For a locus having value 41.6 for a particular subject, who is in liability class 'wide', the entry would be:

```
41.6 wide
```

The symbol `x` is used if the quantitative value or (if there are more than one) liability class is unknown. Hence

```
x x
```

denotes a subject for whom both the value and liability class are unknown.

5.3.4 Named Loci

A named locus is one in which the individual alleles can be identified by some direct method[13] The format for a named subject locus is:

allele_number_1 allele_number_2

Hence for a locus having alleles 1,...,6; a subject having alleles 3 and 4 would be described as:

```
3 4
```

If both the alleles were unknown, the entry would be:

```
x x
```

If only one allele was identified definitely (as 14 say), then the entry would be:

```
14 x
```

If the alleles were labelled '`big`', '`small`' and '`tiny`' then a genotype entry might be:

[13] See also **`read( alsize... );`** in Chapter 7.

```
big small
```

If the locus has been specified as sex-linked, then only one allele should appear in the input pedigree file for male subjects.

5.4 Modifiers

Subjects may be 'tagged' as having special properties by adding particular keywords after their genotypic and phenotypic data.

5.4.1 Loopbreak

Some types of analysis (eg. lodscore calculations) require that any loops within a pedigree should be marked, and that a person be selected at which to 'break' them. This can be done using the **loopbreak** keyword in the subjects description thus:

`1 1 6 7 m` *rest_of_parameters...* **`loopbreak`**

Alternatively, it may be selected by entering the following:

`set loopbreak` `=` *family_name subject_name*`;`

where *fam_name* is the name of a family and *subject_name* is the name of a subject within it.

5.4.2 Proband

The proband for a pedigree may be selected by placing the word **proband** after the input data for one of it's members, thus

`1 1 6 7 m` *rest_of_parameters...* **`proband`**

Alternatively, it may be selected by entering the following:

`set proband` *family_name* = *subject_name*`;`

where *fam_name* is the name of a family and *subject_name* is the name of a subject within it. If no proband is given then *GAS* chooses an individual in each family so as to maximise the speed of computation.

5.5 Validation and Relation

Once all the data is entered, it is analysed to eliminate the vast majority of common genotyping and data-entry errors. However some rare cases of inconsistent inheritance may be missed in large extended pedigrees[14] which contain several fully untyped subjects.

5.5.1 Consistency

Pedigrees are checked to ensure that

[1] parents are of opposite sex

[2] all members of a family are related

[3] each child is consistent with it's parents

[4] each parent is consistent with all it's children

Check [2] can be disabled using the command

```
set checkrelated = n;
```

which may be useful when calculating population based statistics in which the dataset does not consist of complete families. However certain types of analysis (eg. the `lik`... routines) cannot be carried out unless all of the subjects of a family are related to each other via members listed in the pedigree files.

[14] A 'catch-all' routine which performed full genotype elimination for highly polymorphic loci in such families could take months to run!

5.5.2 Loops

Each family is checked for the presence of 'loops' — ie. individuals who are related by more than one pathway of descent (through actions such as incest or multiple marriage).

Any complete loop found is listed, together with suggestions for modifying the pedigree by breaking the loop at certain individuals. By default the program will prompt the user for the name of a subject at which to break each loop, however the command

```
set autoloop = y;
```

will allow *GAS* to automatically break any loops. If any loops are found then a file *loop.gas* will be created listing the chosen breakpoints, and this file may be read into *GAS* the next time the program is run by using

```
read( data loop.gas );
```

after the subject data has been loaded. Loop checking can be disabled by the command

```
set checkloop = n;
```

5.5.3 Unmake

The **lpedigree** format can read data which has been processed using the 'makeped'[15] program by including the parameter **unmake**. Hence to read the file *pedin.dat* which was generated from an l-format file using makeped, enter the following

```
read( lpedigree pedin.dat unmake );
```

Note that if the original pedigree contained loops then *GAS* will attempt to merge the subjects which were duplicated by 'makeped'.

[15] 'makeped' is part of the LINKAGE suite of programs.

5.6 Options for Reading PEDIGREE Data

Two parameters exist which modify the input of family data:

⋆ **locus** load only specified loci

⋆ **overwrite** over-write existing data

The `read( pedigree ... );` command can selectively load subsets of data using the `locus` parameter. For instance, suppose loci `alp`, `bet`, `gam`, `del`, and `eps` are entered into the locus specifications, but only data for the first two are stored in the file *partial.ped*, then[16] use the command

```
read( pedigree partial.ped locus alp bet );
```

which will load in these two loci and leave any values held for the other loci unchanged. The `locus` parameter is also used if the loci are listed in the pedigree file in a different order to that in which they were specified earlier using the `set locus` command.

Normally *GAS* will give an error message if data is read in which would erase previously held information (for instance by loading the same subject twice). If `overwrite` is included in the list of parameters then no message will be given and the second set of data will be replaced by the first.

5.6.1 Automatic Selection of Loci

A g-format pedigree file which has been produced by *GAS* will contain a descriptive line of the form

`pedigree locus` *locus_name(s)...*

GAS will read this and use it to determine which loci are to be loaded from the file (unless the `locus` parameter is used as above, which over-rides this automatic selection). A warning will be given if the pedigree file contains any loci which were not specified earlier using `set locus`.

Example 5.1

The file *autoload.gas* reads the pedigree files *au_even.ped*, *au_odd.ped*, *au_1to5.ped* and *au_6to10.ped*. The combined dataset is written to *au_ped.new*.

[16] As explained in the next section, this is not necessary is the g-format pedigree file was previously created by *GAS*.

Chapter 6 : Control Variables

The behaviour of *GAS* can be altered using a number of control variables, which are modified by using the **set** command in the input gasfile thus:

set *variable_name* = *new_value*;

Most of these variables have default values which are used unless the user tells the program otherwise.

6.1 Output Control

6.1.1 Logfile

The diagnostic output from the program can be written to a file for examination after the run is finished. This will be especially useful if a number of warnings or errors have been produced during the run. A logfile with name *file.ext* can be opened by the command

set logfile = *filename*;

If no extension is present[17] in *filename* then *log* is assumed. Only one logfile can be created per program run, any subsequent **set logfile** commands are noted and ignored.

6.1.2 Outfile

The outfile is used for results produced by the program, and anything produced by the **fprintf** command. To open an outfile give the command

set outfile = *filename*;

If no extension is present, then *out* is assumed. Every time this command is given a new output file is opened (and any previous one is closed). If results are generated before an output file has been opened, then the file **gas.out** is created and used to store the data.

6.1.3 Graphics

Some of the routines in *GAS* provide graphical output. To enable this option, open a file using the command

set psfile = *filename*;

For more details see Chapter 8 on Postscript Graphics later in this manual.

[17] In the name '*fred.dat*' the '*dat*' part is the extension

6.1.4 Allfile

The `allfile` is equivalent to separately setting `logfile`, `outfile` and `psfile` using the same parameter each time. Thus the command

set allfile = *filename*;

opens the files *filename.log*, *filename.out*, and *filename.ps*. Note that the file name parameter must not have an extension or dot '.' within it.

6.1.5 fprintf

The `fprintf` utility can be used to display messages and put additional text into files after the **program** command has been given. **fprintf** uses a syntax based on the ANSI 'C' fprintf routine. The first parameter determines where the message is sent, and must be one (or more) of:

`o`	write to outfile
`l`	write to logfile
`s`	write to screen

Hence the command

```
fprintf( os, "\n\nhello, world" );
```

sends a blank line followed by the text "hello, world" to both the outfile and the screen.

6.1.6 Verbosity

The level of diagnostic information produced by the program (and sent to the screen and logfile) can be modified using the `verbosity` variable thus

`set verbosity =` *level*;

where the *level* is an integer (ie. whole number) in the range 0–3, and larger values produce more information. The default value is 1.

Example 6.1

The gasfile *fpdemo.gas* creates an outfile *fpdemo.out* and logfile *fpdemo.log* and selectively sends information to them and the screen.

6.2 Consistency Checking

6.2.1 Autoloop

The `autoloop` variable determines whether *GAS* automatically breaks any loops it finds. It takes the values `y` or `n`, and has default value `n` so that the user will normally be asked to select subjects for breaking any loops found.

6.2.2 Checkloop

The `checkloop` variable determines whether *GAS* checks each family for the presence of loops. It takes the values `y` or `n`, and has default value `y` so that loops will normally be searched for.

6.2.3 Checkrelated

The `checkrelated` variable determines whether *GAS* checks that all the members of each family are related. It takes the values `y` or `n`, and has default value `y` so that normally every family will be checked to ensure that all of their members are related.

6.2.4 Maxerrors

The `maxerrors` variable determines how many ERROR status messages *GAS* will display before asking if the user wishes to terminate the run. The default value is 5.

6.2.5 Maxwarnings

The `maxwarnings` variable determines how many WARNING status messages *GAS* will display before asking if the user wishes to terminate the run. The default value is 5.

6.3 Loci

6.3.1 Sexlinked

The `sexlinked` command can be used to make loci sex-linked (ie. part of the X-chromosome) by default. Note that this only affects loci declared after it has been altered. The syntax is

```
set sexlinked = y;
```

Using `sexlinked = n;` restores the normal default for any loci declared subsequently.

6.4 Subjects

6.4.1 loopbreak

The `loopbreak` command can be used to designate a particular subject as being a suitable point at which to break a loop. The syntax is

`set loopbreak =` *pedigree_name subject_name*;

6.4.2 proband

The `proband` command can be used to designate a particular subject as being the proband for a pedigree. The syntax is

`set proband =` *pedigree_name subject_name*;

6.4.3 lqunknown

The l-format uses a particular numeric value to denote when the value for a subject at a quantitative locus is unknown (it has the unfortunate default of 0.0). The `lqunknown` variable can be used to alter this value both for reading and writing files. The syntax is

`set lqunknown =` *value*;

so that entering `set lqunknown = -99` will cause *GAS* to mark any l-format subject with quantitative value -99 as being unknown, and similarly any unknown value will be written out as -99 when using `write( lpedigree );`

The categorizing of subjects as having unknown values is done during the `read` process, so subsequently altering `lqunknown` will not affect any subject data that has already been loaded. Hence it is possible to read an lpedigree file then change the value of `lqunknown` and re-write the original file with the new 'unknown' value substituted for the original one.

Chapter 7 : Allele-size Data Input

Automated genotyping using fluorescent markers is becoming increasingly common. The *GAS* program is able to read genotypic data given in terms of the lengths of CA-repeats and to process it into a form suitable for further analysis. The command to read allele-size family data is:

read(alsize *file_name(s)* **locus** *locus_name(s) options(s)...* **);**

where *file_name(s)* are the names of the files containing data, and *locus_name(s)* are the names of the loci to be loaded from them (it is suggested that the real names of marker loci are used rather than some local convention).

GAS incorporates two methods for relating measured lengths to the alleles, called 'fixed' and 'adaptive' binning. With fixed binning the user enters the actual known sizes of all of the alleles when the locus is specified and the new data is categorized according to this.

If the allele sizes are unknown then *GAS* uses an adaptive binning strategy which partitions sizes according to their natural clustering. Some experimental datasets are not sufficiently clear for adaptive global[18] binning, and in these cases adaptive local binning[19] may be used — in which alleles are scored separately within each family (data produced in this latter fashion can be used for linkage studies but not association). The **graph** option may be used to examine the quality of your data — ideally one wishes to see narrow peaks separated by broad empty intervals.

After the data has been read, *GAS* will check it for consistency — ie. that the alleles are sensible, distinct and do not imply illegitimacy. Messages are displayed (and copied to the logfile if you have opened one) which indicate any suspicious data and the action *GAS* has taken to deal with it. All the good data is added to the internal database and may be used in the analysis routines or saved to a file via the **write** command. To correct errors it is necessary to either modify the input file directly, or else to return to the original machine data and re-generate the input file from there after making corrections.

---

[18] We say binning is 'global' if a whole population is scored identically, and 'local' if family-based subsets of the population may be scored differently.

[19] If this is necessary, then the effect of random variations in conditions between runs can be minimized by running all of a family simultaneously on the same gel.

7.1 Input Format

The CA-repeat genotype data should be placed in files having either the following five column format:

locus_name   pedigree   subject   allele1   allele2

or an equivalent four column format:

locus_name   pedigree.subject   allele1   allele2

where *locus_name* is the locus tested. The *pedigree* and *subject* names may be either separated by spaces or a decimal point. *allele1* and *allele2* are the measured lengths of the alleles belonging to the subject at that particular named locus. If data is missing or uncertain, then one or both of the *allele* entries should be replaced by `x`. Data for several loci can appear within the same file.

Example 7.1

The file *als.gas* reads in pedigree, affection-status and some marker locus data from the g-format file *als.ped* and combines this with allele size data for locus 'ca1demo' held in the files *als1.siz* and *als2.siz*. The data is locally binned and written out to the file *alsped.new* in l-format.

7.2 Reading ALSIZE Data

In addition to the **`locus`** specification, there are several optional parameters which alter the criteria by which allele size data is read and binned. Parameters applicable to both global and local binning are:

⋆ **`graph`**	display input data in graphical format
⋆ **`minsize`**	alleles shorter than this are rejected, default 10
⋆ **`maxsize`**	alleles longer than this are rejected, default 1000
⋆ **`overwrite`**	over-write existing data
⋆ **`showspread`**	give statistical information on input data
♡	barcharts drawn with **`graph`** option

if **`overwrite`** is not included, then any existing data is treated as protected and a warning is given if an attempt to over-write it is made.

7.3 Graphical Barchart

The **graph** option will generate barcharts showing the distribution of allele lengths in the input data. By default the barcharts are written in 'block' format to the output file, however if a **psfile** has been set, then a graphical barchart is drawn and subsidiary barcharts are produced showing the characteristics of the alleles in each of the individual input files.

It is highly recommended that the **graph** option is used with each set of incoming data to determine the appropriate method of binning — distinct clusters are suitable for global binning (either fixed or adaptive), whereas a continuous spread can only be scored using the local adaptive algorithm.

The format of the command is

read(alsize *file_names* **locus** *locus_names* **graph** *n***);**

where n is the number of lines used to represent one unit of allele length in the 'block format' text output. If n is not entered, then a default value of 4 is used. Note that the **graph** parameter over-rides all other options, and no binning is performed.

Example 7.2

The file *bar.gas* contains specifications for 2 loci and reads pedigree data from *bin.ped* and size data from *bin1.siz* and *bin2.siz*. Barcharts are created for loci 'alpha' and 'beta', and written in block format to the file *albar.out* and postscript format to *albar.ps*

Printing the results (preferably using a small fixed-width font for the outfile) is useful for examining the distribution across the full range of alleles and hence choosing the optimal algorithm.

7.4 Fixed Global Binning

Fixed global binning is automatically used if the lengths of the alleles were specified when the named locus was described (using **set locus**). This is done (see Chapter 4) using either the **size** and **range** qualifiers or the **minsize** and **maxsize** qualifiers. The size ranges declared during the **set locus** stage may be superceded using:

⋆ **sizerange** this supercedes the value of **range** for all alleles

7.5 Adaptive Binning

Adaptive binning is used if the lengths of the alleles were not specified when the locus was described. The method uses two values **samesize** and **diffsize** to determine which alleles are the same and which are different, according to the criteria:

$\delta l <$ **samesize**	alleles are the same
samesize $\leq \delta l \leq$ **diffsize**	ambiguous, give warning
diffsize $< \delta l$	alleles are different

where δl is the difference in length between two alleles. If global binning is chosen, the data is pre-scanned before binning subjects to determine optimal bins

Optional extra parameters for local binning are:

- ⋆ **diffsize** two alleles differing by more than this are different, default 1.2
- ⋆ **orderfirst** alleles are labelled in the order they are read
- ⋆ **samesize** two alleles differing by less than this are the same, default 0.95

To alter the criteria for local binning, include one or both of the parameters

samesize = a diffsize = b

inside the brackets, where a and b are the desired new values. For instance, the command

```
read( alsize gel1.siz gel2.siz locus d1s20 d1s22
              samesize = 0.8 diffsize = 1.9 minsize = 115
              maxsize = 143 orderfirst );
```

will reject any length shorter than 115 or longer than 143, two alleles less than 0.8 apart will be set as identical, and more than 1.9 will be assumed different, and alleles will be labelled in the order in which they are first encountered when reading the data for each family.

7.5.1 Ambiguous Alleles

Often it will be impossible for *GAS* to decide exactly how to score every single allele. Whenever such ambiguous alleles are found the user is shown the nearest bin(s), together with the distance of the allele from these bin(s), and then presented with a choice of options. These are:

[1] put the ambiguous allele into a nearby bin, which is then stretched to accommodate the allele. Enter the index number of the bin.

[2] make a new bin centred near the allele. Enter '`m`'.

[3] score the allele as unknown. It is marked as `x` in the pedigree and takes no part in subsequent analysis. Enter '`x`'

[4] show the list of bins currently defined. Enter '`s`'.

7.5.2 Bin Display

If the '`s`' option is selected in response to *GAS* finding an ambiguous allele, the list of bins currently defined is displayed in the format:

Bin	Alleles	Diffsize_Limits	Size_of_Bin	Range_of_Alleles
N_B	N_a	D_{min} - D_{max}	S_{min} - S_{max}	R_{min} - R_{max}

These figures have the following interpretation:

N_B	the index of the bin
N_a	the number of alleles currently in the bin
D_{min}, D_{max}	lengths outside this range are put in a different bin
S_{min}, S_{max}	lengths inside this range are put in bin N_B
R_{min}, R_{max}	this is the range of alleles currently in bin N_B

An arrow <- is displayed beside the bin(s) closest to the ambiguous allele, and the user is presented with the list of options again.

7.5.3 Local Adaptive Binning

In local binning each family is scored separately. This is the most robust method for poor quality data, but (since allele '1' may correspond to different physical lengths in different families) it cannot be used for association tests. Local binning is the default for the adaptive method.

7.5.4 Global Adaptive Binning

With global binning, the same bins are used for every family in the dataset. The names and sizes of the bins may be saved (this is necessary if additional people are to be added at a later date) using `write( data ... );`.

To use global binning, include the parameter **global** in the **read** command:

`read( alsize` *parameters...* `global );`

The first stage in adaptive global binning is to pre-scan the data to locate clusters of alleles. These clusters are marked as bins until it is impossible to find a new one which would have at least $2 + \frac{N}{100}$ datapoints in it (where N is the total number of datapoints being read). This cutoff point can be altered using the **minprebin** parameter.

If the final results are unsatisfactory (and you've experimented with a few values of **samesize** and **diffsize**) then you could try manually entering the allele sizes at the earlier **set locus**... stage, using the output from the **graph** command as a guide. If this doesn't work then your only recourse is to switch over to local binning.

7.5.5 Xambiguous

Using the **xambiguous** parameter will cause all ambiguous alleles to be marked as unknown 'x'. However this should be used sparingly as it could wipe out a lot of recoverable data.

Example 7.3

The file *bin.gas* contains specifications for 4 loci and reads pedigree data from *bin.ped* and size data from *bin1.siz* and *bin2.siz*. Locus 'alpha' is binned using the global adaptive algorithm. Locus 'beta' is binned globally using fixed bins (note the specification of the allele lengths in the gasfile) and locus 'gamma' is binned locally. A series of warnings will be produced during the binning process — enter 'c' followed by <return> each time that *GAS* asks if you wish to continue or quit.

The messages are copied into the logfile *bin.log* and the good data is written to *binped.new* for inspection. The new bins created for locus 'alpha' are written to the file *binloc.new*, and may be used for fixed-binning if subsequent data is added.[20]

[20] Note that in this case it may be necessary to add further bins manually if the new dataset contains genuine alleles not present in the initial one. Alternatively both the initial and new size data could be re-loaded simultaneously.

7.6 Consistency Checking

After the allele-size data has been read, *GAS* will check it for consistency — ie. that the alleles are sensible, distinct and do not imply illegitimacy. Messages are displayed (and copied to the logfile if you have opened one) which indicate any suspicious data and the action *GAS* has taken to deal with it. All the good data is added to the internal database and may be used in the analysis routines or sent to a file via the `write` command.

The following checks are performed:

(1) a warning is given if the `alsize` input file contains any subject not listed in the original pedigree data file(s).

(2) a warning is given if the allele lengths are outside of normal limits (the default valid range is 20 to 1000bp).

(3) after labelling the alleles a test is performed to check that each child has a genotype consistent with that of their parents and any full siblings. If an inconsistency is found, the locus data for the whole family containing it is labelled as being unknown.[21]

(4) a warning is issued if the new data will over-write previously loaded data.

(5) with fixed global binning, a warning is signalled if any allele length lies outside the valid known ranges for the alleles. In this case the allele is marked as unknown.

Note from (3) that if even a single bad inheritance is detected within a pedigree then ALL of the members of that pedigree are marked as unknown '`x`' at that locus — the data for the other members will not be accepted until the user has resolved the bad inheritance manually. To correct such errors it is necessary to either modify the input file directly, or else to return to the original machine data and re-generate the input file from there after making corrections.

[21] A common cause of apparent illegitimacy is a heterozygous subject being marked as homozygous by the genotyping program before the data reached *GAS*.

Chapter 8 : Postscript Graphics

Some of the routines in *GAS* are able to display their results graphically. This is done by creating a Postscript file which can be viewed[22] or printed[23] after the *GAS* run has completed. Routines which provide graphical output are marked with the ♡ symbol.

8.1 Output File

The graphical option is selected by specifying a file using **`set psfile`**, thus

`set psfile =` *filename.ext*;

would direct the graphical output into the file *filename.ext*. By default, postscript files produced by *GAS* will be given the extension '*.ps*', so that

`set psfile = project;`

will produce a file called *project.ps*

[22] For instance using 'ghostview' previewer, which is available freely via FTP.
[23] Using a Postscript compatible laserprinter.

8.2 Format Control

The `pscontrol` command can be used to alter the style of the graphical output, for instance to change the size of text or the number of graphs per page. If required this should be done after the `psfile` is selected, and before the `program;` command, thus:

> `set psfile =` *filename*`;`
>
> ⋮
>
> `pscontrol(` *options* `);`
>
> ⋮
>
> `program;`

The available options are:

- ⋆ `clipping` method of line clipping
- ⋆ `displayfile` show the name of the postscript file
- ⋆ `displaygas` show source version of *GAS*
- ⋆ `displaypage` show page number
- ⋆ `displaytime` show time of creating file
- ⋆ `grouping` whether to group related graphs
- ⋆ `showpoints` plot symbols on line graphs
- ⋆ `smoothfactor` interpolation points for smoothing
- ⋆ `smoothing` type of curve smoothing to use
- ⋆ `symbolpoints` type of symbols to use

8.2.1 display

The display options `displayfile`, `displaygas`, `displaypage` and `displaytime` determine how much indentification information is printed on each page. They take the values `y` and `n`, with the default being to show all the information. Thus to prevent the time being shown on each page, include the parameter

`displaytime = n`

8.2.2 clipping

When drawing curves it is possible that the line will go outside of the current graph, and the value of `clipping` determines what to do if this happens.

`clipping`$= 0$	ignore places where line is off graph
$0 <$`clipping`< 1	colour axes grey where line is off graph
`clipping`$= 1$	draw line even if it 'escapes' graph

The default value is 0.5 which makes the axes medium grey if the line goes off the graph (the higher the value of `clipping`, the darker the out-of-range sections will be shaded).

8.2.3 grouping

To reduce paper usage *GAS* will group some types of related graphs on the same page. This feature may be turned off using the parameter

```
grouping = n
```

in which case all the graphs will be expanded to full size and drawn on separate pages.

8.2.4 showpoints/symbolpoints

Curves in *GAS* are drawn from a set of points at which the precise function values are known, and the `showpoints` parameter will cause *GAS* to mark these points with a symbol. The `symbolpoints` parameter allows the user to select the symbol to use (a value of n gives an n-sided polygon, and $-n$ gives an n-sided star). Hence

```
showpoints = y          symbolpoints = 5
```

will draw pentagrams at the points *GAS* is using to plot curves.

8.2.5 smoothing/smoothfactor

All graphics routines draw curves by joining up straight line segments. The default mode in *GAS* (which is recommended as being the most 'honest') is to draw single straight lines between points at which function values are known (**`smoothing`**= 0). However cubic spline interpolation can be used to achieve a more 'rounded' effect by interpolating extra points between the known values; to use this select

```
smoothing = 1
```

The value of **`smoothfactor`** determines the number of intermediate points that the smoothing algorithm will create (default is 5), and higher values give smoother curves, though the size of the postscript file will be increased.

Note: it is possible that the interpolation process may produce curves which 'overshoot'. If you are going to use this option, then you must check (using **`showpoints`**) that this does not happen with your dataset.

Chapter 9 : Creating Subsets

There is often a need to sub-divide the members within a pedigree according to phenotypic or relational criteria. The `select` and `delete` functions provides several options for automatically categorizing subjects and creating new pedigrees from subsets of the original.

The syntax and options for `select` and `delete` are identical, and discussed after the individual routine specifications.

Apology: the `select` and `delete` functions are very powerful, and as a result the format required to control them may at first sight seem confusing. However it is strictly logical, (there are no 'special' cases to learn) and a large number of examples are provided in the text to enable users to modify them as required.

9.1 SELECT

New datasets may be constructed from old ones by creating subsets using the `select` routine. This examines all the individuals in the pedigree according to user-specified criteria and removes those subjects who do not satisfy them. It has the syntax:

`select(` *criteria...* `then` *selectee(s) optional* `);`

The *criteria* is composed of instructions asking *GAS* to count some values relating to a subject and to perform numeric tests on them. If the overall outcome of the tests is true then the *selectee(s)* listed after `then` are marked for selection. After the whole pedigree has been tested, all of the subjects not marked for selection are removed.

If `then...` is not included, the routine assumes that just the subject being tested is to be affected by the result (ie. it is equivalent to entering `then subject`). The optional parameters `makegood` will cause `select` to also remove families which no longer consist of a single group.

9.2 DELETE

New datasets may be constructed from old ones by creating subsets using the `delete` routine. This examines all the individuals in the pedigree according to user-specified criteria and removes those subjects which satisfy them. It has the syntax:

`delete(` *criteria...* `then` *deletee(s) optional* `);`

The *criteria* is composed of instructions asking *GAS* to count some values relating to a subject and to perform numeric tests on them. If the overall outcome of the tests is true then the *deletee(s)* listed after `then` are marked for deletion. After the whole pedigree has been tested, all of the subjects not marked for deletion are retained.

If `then...` is not included, the routine assumes that just the subject being tested is to be affected by the result (ie. it is equivalent to entering `then subject`). The optional parameters `makegood` will cause `delete` to also remove families which no longer consist of a single group.

9.3 Criteria

The first part of both `select` and `delete` is the *criteria* which an expression who's True/False value is to be used as a basis for choosing subjects. Criteria may be simple (composed only of one condition) or compound (made up of several simple criteria tied together using 'and' and 'or').

9.3.1 Simple Criteria

There are two main choices for simple criteria:

[1] `locus` *locus_name*

[2] `relatives`

The test part of each *criteria* is described using one of the operators >, =, <, <=, >= or ! =, followed by a number. For instance, to select all individuals who have qualitative locus *atopy* greater than 3.5 enter

```
select ( locus atopy subject a > 3.5 );
```

9.4 Loci

Selection can be made according to the locus values in categories including **brother**, **child**, **father**, **mother**, **sibling**, **sister**, and **subject**. A full list of the countable relatives can be obtained using **gas help**.

9.4.1 Affection Loci

With affection loci the *value* is either y, n, x, or **liab** *class_name* according to whether we wish to select individuals who are affected, unaffected, unknown-status, or belong to a particular liability class.

* To select all individuals who have at least one affected child

```
select ( locus locus_name child y >= 1 );
```

* To discard all individuals who have unknown status:

```
select ( locus locus_name subject x < 1 );
```

* To select all individuals who belong to liability class 2

```
select ( locus locus_name subject liab 2 > 0 );
```

* To select all individuals who have non-affected mothers

```
select ( locus locus_name mother n > 1 );
```

* To delete all individuals who do not have a father in liability class **bald**

```
delete ( locus locus_name father liab bald < 1 );
```

Example 9.1

The gasfile *sela.gas* reads locus data from *sel.loc* and pedigree data from *sel.ped*, then selects all the subjects who have at least one non-affected child, and writes the new pedigree in g-format to the file *selaped.new*.

Example 9.2

The gasfile *sell.gas* reads locus data from *sel.loc* and pedigree data from *sel.ped*, then selects all the subjects who have a father in liability class **blue** (these fathers also being selected).

9.4.2 Binary loci

There are at present no select options available for binary loci.

9.4.3 Named loci

With named loci the *value* is either **heterozygous**, **homozygous** or allele names (including the 'wildcard' values `matcha` and `matchx`). The `matcha` parameter will match any allele which is not `x`. The `matchx` parameter will match any allele including `x`.

* To select all individuals who have genotype '1 3'

```
select ( locus locus_name subject 1 3 > 0 );
```

* To select all individuals who have a heterozygous father

```
select ( locus locus_name subject heterozygous > 0 );
```

* To delete all individuals who have more than two homozygous children

```
delete ( locus locus_name child homozygous > 2 );
```

* To select all individuals who have genotype '1 x'

```
select ( locus locus_name subject 1 x > 0 );
```

* To delete all individuals who have at least one copy of allele 'b'

```
delete ( locus locus_name subject b matchx > 0 );
```

* To select all individuals who have two known alleles

```
select ( locus locus_name subject matcha matcha > 0 );
```

9.4.4 Quantitative loci

Subjects may be selected or discarded according to the values of quantitative loci. Three values **s**, **a** and **g** may be used for comparison — indicating the smallest, average and greatest values of a set of individuals (for categories with only one individual, such as **mother**, these have identical affect).

* To delete all individuals who have values greater than or equal to 6:

select (locus *locus_name* **subject a** $<$ **6);**

* To select all individuals who have fathers with positive values

select (locus *locus_name* **father a** $>$ **0);**

* To delete all individuals who have at least one child with a value less than 20

delete (locus *locus_name* **child s** $<$ **20);**

* To select all individuals who have no children with a value greater than 20:

select (locus *locus_name* **select g** $<$ **20);**

* To select all individuals who have children with an average value less than 20

select (locus *locus_name* **child a** $<$ **20);**

Example 9.3

The gasfile *selq.gas* reads pedigree data from the file *sel.ped*, selects all the subjects who have a value less than 2.5 at locus 'response', and writes the new pedigree in g-format to the file *selqped.new*.

9.5 Relatives

* Individuals may be selected according to the number of relatives they have in their pedigree (see **gas help** for a full list of the categories of countable relatives). To select individuals who have a father in the pedigree:

```
select ( relative father > 0 );
```

* To delete individuals who do not have a mother in the pedigree

```
delete ( relative mother < 1 );
```

* To select individuals who have at least 3 children

```
select ( relative child > 2 );
```

Example 9.4

The gasfile *selr.gas* reads pedigree data from the file *sel.ped*, selects all the subjects with at least two daughters, and writes the new pedigree to the file *selrped.new*.

9.6 Compound Criteria

The basic commands may be combined to produce more complex effects using the 'and' && and 'or' || operators and appropriate bracketting. Also the expressions used for simple criteria may be compared against each other.

* To select families having discordant sib-pairs (ie. some children affected and some normal) use the following:

select (**locus** *locus_name* **child y** > 0 && **locus** *locus_name* **child n** > 0);

* To delete individuals who have more affected sons than daughters

delete (**locus** *locus_name* **son y** > **locus** *locus_name* **daughter y**);

* To select individuals who have a normal father and affected mother, or vice versa:

select ((**locus** *locus_name* **father n** > 0 && **locus** *locus_name* **mother y** > 0)
|| (**locus** *locus_name* **father y** > 0 && **locus** *locus_name* **mother n** > 0));

* To select individuals for which the sum of two quantitative loci is less than 7.5

select (**locus** *locus_name_1* **subject a** + **locus** *locus_name_2* **subject a** > 7.5);

* To select individuals who are affected at a particular locus and whose children all have a quantitative value greater than 2.5 at a second locus:

select (**locus** *locus_name_1* **subject y** > 0 && **locus** *locus_name_2* **child s** > 2.5);

* To select individuals who are affected at two out of three loci:

select ((**locus** *locus_name_1* **subject y** > 0 && **locus** *locus_name_2* **subject y** > 0)
|| (**locus** *locus_name_2* **subject y** > 0 && **locus** *locus_name_3* **subject y** > 0)
|| (**locus** *locus_name_3* **subject y** > 0 && **locus** *locus_name_1* **subject y** > 0));

Example 9.5

The gasfile *selc1.gas* reads pedigree data from the file *sel.ped*, selects all the subjects affected at locus 'disease' who have at least one son. The new pedigree is written in g-format to the file *selc1ped.new*.

Example 9.6

The gasfile *selc2.gas* reads pedigree data from the file *sel.ped*, selects all the subjects affected at locus disease who have a value either less than 1.5 or greater than 7.0 at locus 'response'. The new pedigree is written in g-format to the file *selc2ped.new*.

9.7 Use of 'then'

The **then** statement allows relatives of the person being tested to be selected or deleted. The list of relatives is the same as that for **select**.

* To select/delete only the subject (this is also done if you omit the **then** part entirely)

```
... then subject
```

* To select/delete both the subject's parents (note that in this case the subject itself will not be acted upon)

```
... then father mother
```

* To select/delete the subject and it's children

```
... then subject child
```

* To select/delete the whole family of the subject

```
... then family
```

Chapter 10 : Editing Data Descriptions

GAS provides facilities for altering the structure and appearance of data. These are grouped under the **edit** command, which has the format

edit(*parameters* **);**

10.1 Lformat

The **lformat** option changes all alphabetic names in the loci and pedigree specification to unique numbers, and writes a list of translations to the logfile. This allows data to be stored in full g-format (ie. using real names for families, diseases and loci) and quickly converted into pure numeric format for running programs such as FASTLINK. The syntax is

edit(lformat);

10.2 Twogen

The **twogen** option re-structures a pedigree into a series of two-generation families. All extended families are split up into a number of nuclear families and given new names. The syntax is

edit(twogen);

10.3 Locus

The **locus** parameter, followed by the name of a locus, has several options:

10.3.1 Nonparam

The command

edit(locus qv nonparam);

will replace the values of quantitative locus **qv** in each subject by their non-parametric rank in the current pedigree.

10.3.2 Delete

The command

`edit( locus lc delete );`

will delete locus `lc` from the current dataset. This will reduce the RAM required by *GAS* and may improve performance on very small machines (it also provides an alternative to using the `locus` parameter with `write( pedigree`... `);` when you only want to write out a subset of the full data).

10.3.3 Makelocal

The command

`edit( locus lc makelocal );`

reduces the number of alleles (of the named locus `lc`) in each pedigree as far as possible, while maintaining uniqueness. This is equivalent to scoring each family locally (see `read(alsize);`).

10.3.4 Rename

The command

`edit( locus la rename lb );`

will rename locus `la` to `lb`.

10.3.5 Updatefreq

The command

`edit( locus la updatefreq` *option* `);`

alters the input allele frequencies for the locus `la` so that they are equal to those found in the dataset. The optional parameters `parent` and `child` set the frequencies to those found in the parental and child populations respectively,[24] otherwise the frequencies found in the whole input dataset are used.

[24] See `dissect( locus );` for a definition of parents and children in this context.

10.3.6 Lumpalleles

The command

edit(locus la lumpalleles *old_alleles...* **into** *new_allele* **);**

Takes the set of alleles listed as *old_alleles...* and renames them all to the single name *new_allele*.

Chapter 11 : Making New Loci

GAS is able to create new trait loci according to the values of existing loci defined at input. For instance, subjects could be defined as being affected by the new trait `atopy` if they were affected at 2 or more of the affection loci `asthma`, `eczema` and `rhinitis`.

This is done using the `set newlocus` command, which has the same format as `set locus` except for the addition of one or more bracketted expressions which describe how the value at the locus is to be calculated.

> `set newlocus` *locus_name type_of_locus*
> *number_of_alleles gene_frequencies...*
> *specifications...*
> (...) ...;

The bracketted expressions have the format

(*item expression_to_calculate*);

where the *item* is the part of the new locus that is to be set, and *expression_to_calculate* is in the same format as that used by `select` and `delete`.

Note that, when calculating a new value, it is important to test that the components are known values, otherwise the results are likely to be incorrect.

11.1 Affection Loci

For affection loci there are 3 choices of *item*. These are:

`x`	set status as unknown if true
`y`	set status as affected if true
`n`	set status as not-affected if true

If the (`x` ...) bracket is present and true for a particular subject then the (`y` ...) and (`n` ...) terms are ignored. The (`x` ...) term cannot be used alone.

If both the (`y` ...) and (`n` ...) terms are present, but neither is true, then the value is taken to be unknown. The (`y` ...) term is always evaluated before the (`n` ...) term, and if the former is true, then the latter is ignored.

11.2 Quantitative Loci

For quantitative loci there are 2 choices of *item*. These are:

`x`	set status as unknown if true
`q`	set value to numeric expression

If the `( x ... )` bracket is present and true for a particular subject then the `( q ... )` term is ignored. The `( x ... )` term cannot be used alone.

The `( q ... )` term may use any of the mathematical functions built into *GAS* (logarithms, exponentials, '+', '-', etc.)[25] to combine and transform the values of loci and relative counts.

Example 11.1

The gasfile *makenew.gas* reads pedigree data from the file *skin.ped*, and creates four new loci. Subjects are assumed to be affected at locus 'spotty' if they are affected at either of loci 'acne' or 'pimples'. Subjects are assumed to be affected at locus 'very_spotty' if they have a value greater than 5 at quantitative locus 'spots_per_cm2', or if they are affected at locus 'boils'. Locus 'logspotty' is set to be the logarithm of the value at locus 'spots_per_cm2' (note the test to avoid zero arguments). The new locus specifications are written to the file *newskin.loc* and the modified pedigree is saved in the file *newskin.ped*.

[25] Use **`gas help`** for a list of the mathematical functions

Appendix C : Control Variables

Below is a summary of the *GAS* control variables listed elsewhere in this manual. The '*' denotes that there is no default value for a particular item, and that it will not be used unless explicitly set by the user.

variable	function	values	default
autoloop	automatically break loops	y/n	y
checkloop	check for loops	y/n	y
checkrelated	check all of pedigree is related	y/n	y
logfile	name of logfile	*	*
loopbreak	break loop at location	pedigree subject	
lqunknown	quantitative unknown-code for l-format	$\pm\infty$	0
maxerrors	maximum number of errors	1–∞	5
maxwarnings	maximum number of warnings	1–∞	5
outfile	name of output file	*	*gas.out*
proband	designate proband for a family	pedigree subject	
sexlinked	X-chromosome data	y/n	n
verbosity	level of information to screen/logfile	0–3	1

The **gas help** facility can be used to produce a list of these variables.

Appendix I : Installing GAS

The full package for *GAS*2.0 consists of 2 Postscript manuals, an executable file (the actual *GAS* program), and a collection of example files. The manuals are stored uncompressed in plain Ascii format, and should be sent directly to a Postscript laserprinter. Some of the executable and example files have been compressed into single packages for transfer, and full instructions for restoring them are given below.

I.1 IBMPC DOS

The program and all the examples are contained in the file gas20dos.exe, and once you've put this in a local directory you should run it (under DOS) by typing

```
gas20dos
```

which will extract everything into your current directory. Note that the *GAS* program runs under DOS rather than Windows, and that the files *32RTM.EXE* and *DPMI32VM.OVL* (which are supplied with the package) must be in the DOS path in order to run *GAS*. Furthermore, the DOS version of *GAS* requires that the Windows directory must also be in the DOS path.

I.2 Unix Systems

The executables for different Unix systems are each stored in compressed tar files with names *gas20_X.tar.Z* where '*X*' identifies the the platform. The example files are stored in the single file *gas20_examples.tar.Z* which is common to all Unix platforms.[26] To extract the program and examples use the command:

zcat *filename* | tar –xvf –

on all the '*tar.Z*' files you've retrieved.

[26] The files were archived with the command: tar –cvf – * | compress > *bundlefile*

I.3 VMS

The VMS executables are stored without any compression or processing – simply download, rename to *gas.exe*, and run. To use *GAS* from the command line you need to rename the *gas20_computer.exe* file to *gas.exe* and include the following in your login.com file:

$$\text{gas :== \$}DIRECTORY\text{gas.exe}$$

where *DIRECTORY* is replaced by the directory (and full disk specification) in which the *gas.exe* program is placed.

There is no special example package for VMS, so you should load either the DOS or Unix package as described above (the contents of the two are identical).

End of Gas Manual version 2.0