

Importing and simulating data

One of the more frustrating tasks associated with the use of any data analysis software concerns the importation of data. Data can be imported into R/qtl in a variety of formats, but users often have trouble with this step. In this chapter, we describe how to import QTL mapping data into R for use with R/qtl. We further discuss the simulation of QTL mapping data. In an optional section, we describe the internal format that R/qtl uses for QTL mapping data.

As this may be the reader's first exposure to R, we will introduce some of the basic aspects of R as we go along. We should again emphasize that the novice user will benefit by spending a couple of days reading Dalgaard (2002) and playing with R.

Before you do anything, you must install R and the R/qtl package; this is described in Appendix A. After invoking R, you must type `library(qtl)` to load the R/qtl package. (In R, R/qtl is known as the `qtl` package or library.) It is best to create a `.Rprofile` file containing this command, so that the package will automatically be loaded whenever you invoke R. (See Sec. A.3.)

Essentially all tasks in R are performed via *functions*, such as the `library` function mentioned above. Appendix B contains partial list of the functions in R/qtl. A complete list may be viewed by typing the following.

```
> library(help=qtl)
```

The `>` symbol is the R prompt, which you will observe when R is ready to accept input commands. R commands may be spread over several lines, in which case the R prompt turns into the `+` symbol, indicating a continuation line. (Appearance of the `+` prompt when one believes one's command is complete may indicate imbalance in parentheses. Press the escape key to cancel the command.) R input will be shown in a *slanted typewriter font*, while output will be in a *plain typewriter font*. (The output for the above command was suppressed, as it would fill a couple of pages.)

Note that the up and down arrow keys may be used to scroll back through previously entered commands. Emacs users will be pleased to find that many

of the Emacs key bindings may be used. (But be careful about Ctrl-p, which may lead you to print a page.)

2.1 Importing data

Importing QTL mapping data into R is accomplished with the `read.cross` function. Data may be read in a variety of formats. We strongly recommend the comma-delimited formats discussed in the next subsection, but Map-Maker/QTL, Map Manager QTX, and QTL Cartographer formats may also be used. Sample data files in most of the formats are available at the R/qtl web site ([http://www.rqtl.org/sampledata](http://www.rqtl.org/sampleddata)). The help file for `read.cross` contains the complete details on the file formats and the use of the function. The help file may be viewed by typing `?read.cross`; see Sec. A.5. Note that basic use of the `read.cross` function is described in Sec. 2.1.1 on the comma-delimited formats and is not repeated in the subsections on the other formats.

Before contemplating loading one's data into R, it must be assembled into one of the accepted formats. While the comma-delimited formats can be created in OpenOffice, Microsoft Excel, or other spreadsheet programs, a different format (or computer program) might be best for entering the data into the computer. (And ideally data should enter the computer directly from the measurement device, rather than be input by hand.) The reformatting of data files to conform to the requirements of specific software is a frequent task for geneticists, and hand manipulation of data files is time-consuming and error-prone. Thus we recommend that geneticists learn to program in a language like Perl, which will greatly simplify the task. While the up-front investment to learn Perl is large, the value such knowledge will provide over one's career is far larger.

2.1.1 Comma-delimited files

The recommended format for QTL mapping data to be imported into R/qtl is the comma-delimited format, "`csv`" (an abbreviation of "comma-separated values"). Several variations on this format will be described below. We begin by discussing the basic one.

In the basic "`csv`" format, all phenotype and genotype data, plus the genetic map of the typed markers, are combined into a single file with fields delimited by commas. The file may be constructed in a spreadsheet, such as OpenOffice or Microsoft Excel; an example is illustrated in Fig. 2.1. Be careful about the use of commas within the fields (though the use of quotation marks should prevent this from being a problem).

The initial columns are phenotypes (at least one phenotype must be included, such as a numeric index for each individual). Subsequent columns are markers. The first row contains the phenotype and marker names. The second

	A	B	C	D	E	F	G	H	I	J
1	pheno	sex	pgm	c1m1	c1m3	c1m4	c1m5	c2m1	c2m2	c2m3
2				1	1	1	1	2	2	2
3				8.3	49.0	59.5	89.0	1.0	15.0	45.0
4	0.093	f	0	B	B	–	H	B	B	B
5	0.177	f	0	H	H	H	H	H	H	H
6	–	f	0	H	B	A	A	H	–	B
7	0.230	f	0	B	H	H	–	A	A	A
8	0.228	f	0	B	–	H	H	H	B	B
9	0.279	f	0	B	B	A	H	A	H	H
10	0.419	f	0	H	H	H	H	A	B	B
11	0.427	f	0	–	A	B	B	B	H	H
12	0.282	f	0	–	A	B	B	A	A	A
13	0.400	f	0	H	B	A	A	–	H	H
14	0.521	f	0	B	H	B	B	H	H	H
15	0.385	f	0	H	B	A	A	B	H	–
16	0.518	f	0	–	H	H	H	H	H	B

Figure 2.1. Part of a data file in the "csv" format, as it might be viewed in a spreadsheet.

row must have empty fields in each of the phenotype columns. (This is quite rigid; even a space character will mess things up.) For the genotype columns, the second row should contain chromosome assignments. Numbers are best; character strings, such as "Chr 1" or "six" will make later data manipulation more cumbersome. Use "X" or "x" to identify the X chromosome.

An optional third row can contain the centiMorgan (cM) positions of the genetic markers. The fields in the phenotype columns should again be blank. Marker order is taken from the cM positions, if provided; otherwise it is taken from the column order.

Subsequent rows correspond to the individuals, with phenotypes followed by genotypes. Missing data should be indicated by "NA" or "–" or some other code. (It is always best to insert some code indicating missingness rather than leave some cells empty, as empty cells can be ambiguous: was the value missing or was an error in data entry made?) Multiple missing data codes may be used, but consistency between the phenotype and genotype data is required: a missing value code for the genotype data cannot be a legitimate phenotype and vice versa. No missing values are allowed in the chromosome identifiers or genetic map positions.

For a backcross, two genotype codes are to be used: one for homozygotes (e.g., AA) and one for heterozygotes (AB). For an intercross, five genotype codes may be used: the two homozygotes (AA and BB), the heterozygote (AB), and two further genotype codes to be used for dominant markers, such as D for "not BB" (i.e., AA or AB) and C for "not AA" (i.e., AB or BB), as used by the MapMaker software.

Consistency in genotype codes is required: one cannot use both A and AA to indicate a homozygous A genotype. Also note that spaces can mess things

Table 2.1. Possible intercrosses, and the appropriate code for the `pgm` “phenotype.” In the crosses, females are always listed first, so $A \times B$ means a female A crossed to a male B.

Cross	Possible genotypes		pgm code
	Females	Males	
$(A \times B) \times (A \times B)$	AA, AB	A-, B-	0
$(B \times A) \times (A \times B)$	AA, AB	A-, B-	0
$(A \times B) \times (B \times A)$	AB, BB	A-, B-	1
$(B \times A) \times (B \times A)$	AB, BB	A-, B-	1

up: “A ” is treated as different from “A”. It is best to ensure that there are no spaces in the final data file.

X chromosome genotypes should be coded just like the autosomal genotype data; in particular, hemizygous males should be coded as if they were homozygous, rather than using separate codes for hemizygous and homozygous genotypes. If X chromosome genotype data are included, one of the phenotypes should indicate the sex of the individuals. This may be called “sex” or “Sex,” and the sexes may be coded by 0/1 for females/males, or by the codes `f/m`, `F/M`, or `female/male`.

Further care is required for the X chromosome genotype data in an intercross, as the direction of the cross must be known. Four possible intercrosses may be performed, as shown in Table 2.1. In all cases, the males are hemizygous A or B at any one locus, but in the crosses $(A \times B) \times (A \times B)$ and $(B \times A) \times (A \times B)$, the females are either AA or AB, while in the crosses $(A \times B) \times (B \times A)$ and $(B \times A) \times (B \times A)$, the females are either AB or BB. Thus, the order of the cross producing the F_1 male is critical; for example, we wish to know whether the paternal grandmother was from strain A or B.

We thus require, for intercrosses, a “phenotype” column named `pgm` (for “paternal grandmother”), with codes 0 and 1 indicating which individuals came from which cross, as shown in Table 2.1.

If one includes a phenotype named “id,” “ID,” or “Id,” it will be assumed to provide individual identifiers. These will be used in certain places to indicate the individuals (such as in `plot.geno`; see Sec. 3.5).

The specification of a file in the “`csv`” format is now complete. If the file was created in a spreadsheet program, such as OpenOffice or Microsoft Excel, you will need to use “Save as” and select the format “CSV (comma-delimited)” to create the actual file. The result will look something like that shown in Fig. 2.2. A complete example is provided at the R/qlt web site.

With our first file format understood, we now turn to the use of `read.cross` to load the data into R.

A list of the input arguments for `read.cross` may be viewed via the `args` function, as follows. (We often use `args` to get a quick reminder of the input to a function.) Remember that, if R/qlt is not yet loaded, one must use

```

pheno,sex,pgm,c1m1,c1m3,c1m4,c1m5,c2m1,c2m2,c2m3,c2m4,...
,,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,4,4,4,4,5,5,5,5,...
,,8.3,49,59.5,89,1,15,45,68.9,80.9,87.4,99,0,11.2,39....
0.093,f,0,A,B,A,A,B,H,H,H,H,H,B,H,H,B,B,H,A,A,A,A,H,...
0.177,f,0,B,H,H,H,H,B,B,B,B,H,H,H,-,H,H,H,H,A,H,A,...
0.271,f,0,B,A,H,H,H,H,H,A,A,A,A,H,H,H,H,H,B,-,B,B,H,...
0.230,f,0,B,B,A,H,B,B,B,B,B,H,B,A,H,H,B,B,H,H,B,B,H,...
0.228,f,0,H,H,H,H,H,H,H,H,B,B,B,B,H,H,H,H,B,H,H,H,B,...
0.279,f,0,H,B,A,A,H,B,B,H,A,A,-,A,A,A,H,H,H,A,A,H,B,H,...
0.419,f,0,B,H,H,H,A,A,A,A,A,A,B,B,B,B,B,B,H,H,H,H,...
0.427,f,0,B,A,H,H,H,B,B,B,B,B,B,H,H,H,B,B,B,H,H,A,A,...
0.282,f,0,B,B,A,H,A,H,H,A,A,A,B,B,H,A,-,B,H,H,H,H,B,...
0.4,f,0,H,H,H,H,A,B,B,H,H,H,H,H,H,B,H,H,H,H,H,H,...
0.521,f,0,H,A,B,B,B,H,H,H,H,H,H,H,A,A,A,H,B,B,B,H,H,...
0.385,f,0,A,A,B,B,A,A,A,H,H,H,H,B,B,H,H,H,H,H,A,A,A,...
0.518,f,0,H,B,A,A,H,H,H,B,B,B,A,A,H,H,A,H,H,H,H,H,...
:
:

```

Figure 2.2. Part of a text file in the "csv" format. The terminal dots in each line are just to indicate that the file extends quite far to the right.

the `library` function to make it available. (Ignore the `NULL`; that's just a meaningless bit from the `args` function.)

```

> library(qtl)
> args(read.cross)

function (format = c("csv", "csvr", "csvs", "csvsr", "mm",
  "qtx", "qtlcart", "gary", "karl"), dir = "", file, genfile,
  mapfile, phefile, chridfile, mnamesfile, pnamesfile,
  na.strings = c("-", "NA"), genotypes = c("A", "H", "B", "D",
  "C"), alleles = c("A", "B"), estimate.map = TRUE,
  convertXdata = TRUE, ...)
NULL

```

This is, admittedly, rather forbidding, but not all of the arguments will be needed in all cases. Note that the `c` function is used to combine multiple items together into a vector.

The argument `format` will be used to indicate that we are reading data in the "csv" format. The possible formats are shown; the first listed is taken as the default. The argument `dir` is used to indicate the directory in which the file appears. By default, it is assumed that the file is in the current working directory. (For details on how to select or change the working directory, see

Sec. A.4.) The argument `file` will be used to give the name of the data file. The other file arguments are used for formats in which the data are split across multiple files.

The argument `na.strings` is used to indicate the set of missing data codes. By default, either “-” or “NA” will be treated as missing. Note that most things are case-sensitive, so “na” will be treated as different from “NA” and “Na”. If all of these appear in the data file, all should be indicated via the `na.strings` argument.

The argument `genotypes` is used to indicate the genotype codes, and takes a vector of character strings. The order of the codes in the string is important. We often forget whether “D” stands for “not AA” or “not BB,” and so we generally must refer to the help file for `read.cross`, where this is explained. Note, again, that the codes are case-sensitive, so “a” will be treated as different from “A.”

The argument `alleles` is used to indicate custom names for the alleles (single-character names are best), so that if one does a mouse cross of BALB/c $\times$ DBA/2, one might want to use the codes C and D for the alleles. These will be used in certain plots (such as of phenotype against genotype) and summaries.

If the genetic map positions of the markers are not provided in the file and `estimate.map=TRUE`, the intermarker distances will be estimated, while if `estimate.map=FALSE`, a dummy map will be created. (If genetic map positions *are* provided, this argument will be ignored.) Estimation of the genetic map can sometimes be time-consuming, and so one may wish to use `estimate.map=FALSE`. One may later estimate the map with the function `est.map` and plug it into the data object with `replace.map`; see Sec. 3.4.3.

If marker positions are provided in the file, it is important that no two markers are placed at precisely the same position. If they are, this may be rectified with the function `jittermap`; see page 84.

The “...” at the end of the specification of `read.cross` is used to allow additional arguments to be specified; these are passed to the more basic R function `read.table`, which does the actual work of reading in the data. Its use will be explained further below.

There seems a lot to understand, but use of `read.cross` is generally not so tedious as it might appear, as most of the arguments to the function can be ignored. For example, suppose the data in Figures 2.1 and 2.2 are saved in one’s working directory as the file `mydata.csv`. One could read this into R with the following.

```
> mydata <- read.cross("csv", "", "mydata.csv")
```

Note that “<-” is the *assignment operator*. The data are read from the `mydata.csv` file and combined into a single object (with a very special internal format, described in Sec. 2.6.1) and assigned to `mydata`. This will be a new object in our R workspace that we may manipulate and analyze. Type `ls()` or `objects()` to list the objects in your workspace.

Also note that arguments to functions in R may be specified by their position in the list, by their name, or they may be left unspecified (in which case

the default values are assumed). Thus, in the code above, it is assumed that `format="csv"`, `dir=""`, and `file="mydata.csv"`, and we need not specify values for `na.strings`, `genotypes`, or `alleles`, as the default values suffice for our data. All of the following lines of code are equivalent.

```
> mydata <- read.cross("csv", , "mydata.csv")
> mydata <- read.cross("csv", file="mydata.csv")
> mydata <- read.cross(format="csv", file="mydata.csv")
> mydata <- read.cross(file="mydata.csv", format="csv")
> mydata <- read.cross(file="mydata.csv")
```

If the data file were in some location other than the R working directory, we would need to specify its location with the `dir` argument. The directory (or folder) hierarchy is indicated with forward slashes (/). In Windows, it is traditional to use backslashes (\), but these will not work in R, though double-backslashes (\\) may be used in place of forward slashes.

For example, if we were working on a Macintosh and our file was on the Desktop, we might use the following code. The tilde (~) denotes our home directory.

```
> mydata <- read.cross("csv", "~/Desktop", "mydata.csv")
```

If we were working in Windows and the file was located in `c:\My Data`, we could use the following code.

```
> mydata <- read.cross("csv", "c:/My Data", "mydata.csv")
```

If we had coded the genotype data differently, we would need to use the `genotypes` argument. Because of all of the intervening file name arguments, the `na.strings`, `genotypes`, and `alleles` arguments generally must be specified by name. For example, suppose missing data were coded “na” and that the genotypes were coded BB/BC/CC. Then the data would be read as follows.

```
> mydata <- read.cross("csv", "", "mydata.csv", na.strings="na",
+                       genotypes=c("BB", "BC", "CC"),
+                       alleles=c("B", "C"))
```

We recommend downloading the example “csv” data file (`listeria.csv`) from the R/qtl web site and trying to load it into R. (The file is included with the R/qtl package, but it is in a spot that may be difficult to find.) If one has trouble importing one’s own data, it is a good idea to try importing a file that is known to be correct, so one may determine whether the problem concerns some incompatibility in the file or an incomplete understanding of the use of `read.cross`.

Outside the United States, commas are sometimes used instead of periods in numbers, and so semicolons are sometimes used instead of commas in such CSV files. Files of this sort may also be read; one must make use of the flexibility in the `read.cross` function through the “...” in its specification, through

```

# The ch3c data
# File created by Karl W Broman, 7-19-06
# Intercross between C57BL/6J and A/J
# 100 females from the cross (AxB)x(AxB)
# 101 markers, including 10 on the X chromosome
pheno,sex,pgm,c1m1,c1m3,c1m4,c1m5,c2m1,c2m2,c2m3,c2m4,...
,,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3,4,4,4,4,5,5,5,5,...
,,8.3,49,59.5,89,1,15,45,68.9,80.9,87.4,99,0,11.2,39,...
0.093,f,0,A,B,A,A,B,H,H,H,H,H,B,H,H,B,B,H,A,A,A,A,H,...
0.177,f,0,B,H,H,H,H,B,B,B,B,H,B,H,H,H,-,H,H,H,H,A,H,A,...
0.271,f,0,B,A,H,H,H,H,H,A,A,A,A,H,H,H,H,H,B,-,B,B,H,...
0.230,f,0,B,B,A,H,B,B,B,B,B,H,B,A,H,H,B,B,H,H,H,B,B,H,...
0.228,f,0,H,H,H,H,H,H,H,H,B,B,B,B,H,H,H,H,B,H,H,H,B,...
0.279,f,0,H,B,A,A,H,B,B,H,A,A,-,A,A,A,H,H,H,A,A,H,B,H,...
0.419,f,0,B,H,H,H,A,A,A,A,A,A,B,B,B,B,B,B,B,H,H,H,H,...
0.427,f,0,B,A,H,H,H,B,B,B,B,B,B,H,H,H,B,B,B,H,H,A,A,A,...
:
:

```

Figure 2.3. An example file in the "csv" format with comment lines included.

which further arguments are passed down to the more basic `read.table` function. That function allows arguments `sep`, for specifying the field separator, and `dec`, for specifying the character used for the decimal point.

Thus, if the `mydata.csv` file had used semicolons and commas rather than commas and periods, we would read it into R with the following code.

```
> mydata <- read.cross("csv", , "mydata.csv", sep=";", dec=",")
```

Note that these additional arguments *must* be specified by name.

One may include comments in an input file, to be ignored when it is imported, but useful to document its contents. A single symbol, such as `#`, may be used to indicate that the remainder of the line is to be ignored. The chosen symbol cannot appear anywhere in the data, and is indicated, in the call to `read.cross`, via the `comment.char` argument. (In R versions 2.3.1 and earlier, `comment.char="#"` was the default, but in R versions 2.4.0 and later, the default has become `comment.char=""`, and so no such commenting character is assumed.)

For example, the file in Fig. 2.3 contains initial comment lines, indicated by `#`. To read this file into R, we would use the following code.

```
> mydata <- read.cross("csv", , "mydata.csv", comment.char="#")
```

There are three related comma-delimited formats: "`csvr`", "`csvs`", and "`csvsr`". These are primarily for the case of expression genetic data, in which

	A	B	C	D	E	F	G	H	I	J
1	pheno			0.093	0.177	–	0.230	0.228	0.279	0.419
2	sex			f	f	f	f	f	f	f
3	pgm			0	0	0	0	0	0	0
4	c1m1	1	8.3	B	H	H	B	B	B	H
5	c1m3	1	49.0	B	H	B	H	–	B	H
6	c1m4	1	59.5	–	H	A	H	H	A	H
7	c1m5	1	89.0	H	H	A	–	H	H	H
8	c2m1	2	1.0	B	H	H	A	H	A	A
9	c2m2	2	15.0	B	H	–	A	B	H	B
10	c2m3	2	45.0	B	H	B	A	B	H	B
11	c2m4	2	68.9	B	H	H	A	B	A	H
12	c2m5	2	80.9	B	B	A	A	B	A	H
13	c2m6	2	87.4	H	B	A	A	B	A	H
14	c2m7	2	99.0	B	B		–	B	A	H
15	c3m1	3	0.0	A	B	A	B	H	B	H
16	c3m2	3	11.2	H	B	–	B	H	B	H

Figure 2.4. Part of a data file in the "csvr" format, as it might be viewed in a spreadsheet.

QTL mapping is to be performed with the expression of all genes on a microarray, so that one has thousands or tens of thousands of phenotypes.

The "csvr" format is just like the "csv" format, but with rows and columns interchanged. (The "r" is for *rotate*, but the file is technically *transposed* rather than rotated.) In Fig. 2.4, the file from Fig. 2.1 is shown in the "csvr" format. All other aspects are the same as before, and the use of `read.cross` is unchanged, so such a file (call it "mydata_rot.csv") could be read in as follows.

```
> mydata <- read.cross("csvr", , "mydata_rot.csv")
```

Of course, other arguments, such as `genotypes`, may be used as before.

The "csvs" format is similar to the "csv" format, but with separate files for the phenotypes and the genotypes. The genotype data file must begin with a single column containing individual identifiers, followed by columns for each of the markers. As with the phenotype columns for the "csv" format, this initial column must have empty cells in the rows for the chromosome assignments and marker positions. The phenotype data file must contain a column with precisely the same name and contents, so that we can be sure that the phenotype and genotype data are appropriately aligned. An example of this format is display in Fig. 2.5.

To read data in the "csvs" format, one must specify the names of both files. This may be done via the `read.cross` arguments `genfile` and `phefile`, as follows. (We assume that both files are in the current working directory.)

```
> mydata <- read.cross("csvs", genfile="mydata_gen.csv",
+                       phefile="mydata_phe.csv")
```

Phenotype data file					Genotype data file					
	A	B	C	D		A	B	C	D	E
1	pheno	sex	pgm	id	1	id	c1m1	c1m3	c1m1	c1m3
2	0.093	f	0	1	2		1	1	1	1
3	0.177	f	0	2	3		8.3	49.0	8.3	49.0
4	–	f	0	3	4	1	B	B	B	B
5	0.230	f	0	4	5	2	H	H	H	H
6	0.228	f	0	5	6	3	H	B	H	B
7	0.279	f	0	6	7	4	B	H	B	H
8	0.419	f	0	7	8	5	B	–	B	–
9	0.427	f	0	8	9	6	B	B	B	B
10	0.282	f	0	9	10	7	H	H	H	H
11	0.400	f	0	10	11	8	–	A	–	A
12	0.521	f	0	11	12	9	–	A	–	A
13	0.385	f	0	12	13	10	H	B	H	B

Figure 2.5. Part of the genotype and phenotype data files for an example of the "csvs" format, as they might be viewed in a spreadsheet.

For the user's convenience, if the `phefile` argument was not specified, but the `file` and `genfile` arguments were, we assume that `file` and `genfile` are indicating the genotype and phenotype data files, respectively. This can simplify the code a bit. For example, suppose that we are working in a directory `MyProject/R`, and that the two data files are sitting in the directory `MyProject/Data`. The data could be imported as follows.

```
> mydata <- read.cross("csvs", "../Data", "mydata_gen.csv",
+                      "mydata_phe.csv")
```

The "csvsr" format is just like the "csvs" format, but with both files rotated as in the "csvr" format. We use `read.cross` in the same way as for the "csvs" format.

2.1.2 MapMaker/QTL

The format "mm" is for data in the format used by the MapMaker software. There are two files, a `.raw` file containing the genotype and phenotype data and a second file containing the genetic map information. Examples of these files are provided on the R/qtl web site.

The genetic map file may be in one of two formats. First, one may use a `.maps` file, produced by MapMaker/Exp. Second, one may create a space-delimited file, as illustrated in Fig. 2.6, with one row for each marker. The first column is the chromosome assignment, the second column is the marker name (which must match that used in the `.raw` file exactly), and an optional third column may contain the cM position of each marker.

Use of `read.cross` to read data in the "mm" format is similar to the case of the "csvs" format, discussed in the previous subsection. Specify the `.raw` file

```

1 D10M44 0.00
1 D1M3 1.00
1 D1M75 24.85
1 D1M215 40.41
1 D1M309 49.99
1 D1M218 52.80
1 D1M451 70.11
1 D1M504 70.81
1 D1M113 80.62
1 D1M355 81.40
1 D1M291 84.93
1 D1M209 92.68
1 D1M155 93.64
2 D2M365 0.00
2 D2M37 27.94
2 D2M396 47.11
:
:
```

Figure 2.6. The initial portion of a space-delimited file that may be used to indicate marker locations for the MapMaker ("mm") format.

with the `file` argument and the genetic map file with the `mapfile` argument. (The format of the genetic map file is determined automatically.) Note that the `na.strings` and `genotypes` arguments are ignored with this format, as such codes are specified within the `.raw` file.

For the user's convenience, if the `mapfile` argument was not specified, but the `genfile` argument was, we assume that `genfile` indicates the genetic map file. This can simplify the code a bit. For example, suppose that we are working in a directory `MyProject/R`, and that the two data files are sitting in the directory `MyProject/Data`. The data could be imported as follows.

```
> mydata <- read.cross("mm", "../Data", "mydata.raw",
+                      "mydata.maps")
```

2.1.3 QTL Cartographer

The format "qtlcart" is for data in the format used by the QTL Cartographer software. There are two files, a `.cro` file containing the genotype and phenotype data and a `.map` file containing the genetic map. Examples of these files are provided on the R/qtl web site.

We use `read.cross` to read the QTL Cartographer files in a manner similar to that used for the MapMaker files. For example, suppose we are working in

a directory `MyProject/R` and that the two data files are in the directory `MyProject/Data`; they could then be imported as follows.

```
> mydata <- read.cross("qtlcart", "../Data", "mydata.cro",
+                      "mydata.map")
```

2.1.4 Map Manager QTX

The format `"qtx"` is for data in the format used by the Map Manager QTX software. There is a single file, generally with extension `.qtx`, containing all of the genotype and phenotype data as well as the genetic markers' chromosome assignments and order. Genetic map positions for the markers are generally not included in the file, and so must be estimated. An example file is provided on the R/qtl web site.

Loading data from a `.qtx` file into R/qtl is simple. The `na.strings` and `genotypes` arguments need not be used, as such codes are included within the file. Suppose that we are working in the directory `MyProject/R`; to read the `mydata.qtx` from the directory `MyProject/Data`, type the following.

```
> mydata <- read.cross("qtx", "../Data", "mydata.qtx")
```

As the genetic map positions for the markers are generally not provided in the `.qtx` file, and so must be estimated from the data, the import of the data can be time consuming. One may wish to use `estimate.map=FALSE` in the call to `read.cross`, and then use `est.map` and `replace.map` to estimate the map and then plug it into the data. This process is described in more detail in Sec. 3.4.3, but let us briefly consider a simple example.

```
> mydata <- read.cross("qtx", "", "mydata.qtx",
+                      estimate.map=FALSE)
> themap <- est.map(mydata, error.prob=0.001)
> mydata <- replace.map(mydata, themap)
```

In the first line of code, we read in the data without estimating the intermarker distances, and so a dummy map is inserted into the `mydata` object. In the second line, we call `est.map` to estimate the genetic map, here assuming that genotypes may be in error with probability 0.1%. The result is placed in the object `themap`. In the final line of code, the `replace.map` function is used to replace the map within `mydata`, inserting `themap` in its place. The output is the same data but with a different map; we assign it back to `mydata`, writing over the original data. (We might have assigned it to an object with a different name, in which case both would appear in our R workspace.)

2.2 Exporting data

Data may be exported from R/qtl into several formats. This may be useful, for example, if one wishes to compare results from R/qtl to those from QTL Cartographer, or simulate data in R/qtl and analyze them in Cartographer.

The `write.cross` function is used for this purpose. The `cross` argument is the cross object to be exported. The `chr` argument may be used to indicate a subset of chromosomes that should be exported. The `format` argument indicates the format to which the data should be written.

The `filestem` argument indicates the initial part of the file names. For example, with the `qtlcart` format, `.cro` and `.map` files will be created. If one uses `filestem="mydata"`, the files `"mydata.cro"` and `"mydata.map"` will be created.

The `filestem` can include a directory, so that the files may be written somewhere other than the current working directory. For example, if one wishes to save chromosomes 5 and 13 of the `listeria` data to a file in the `"csv"` format on the Desktop on a Macintosh computer, use the following code.

```
> data(listeria)
> write.cross(listeria, "csv", "~/Desktop/listeria", c(5, 13))
```

2.3 Example data

A variety of example data sets are included with R/qtl. A complete list may be obtained with the following.

```
> data(package="qtl")
```

Of particular interest are the `hyper` and `listeria` data, which will be used as the main examples in this book.

The `hyper` data set is from Sugiyama *et al.* (2001). (It was also discussed in Sec. 1.2.) This is a backcross using the C57BL/6J and A/J inbred mouse strains, with the F₁ mated back to the C57BL/6J strain. There are 250 male backcross individuals. Mice were given water containing 1% NaCl for two weeks; the phenotype is blood pressure (actually the average of 20 blood pressure measurements from each of 5 days).

The `listeria` data set is from Boyartchuk *et al.* (2001). This is an intercross using the C57BL/6ByJ and BALB/cByJ inbred mouse strains. There are 120 female intercross individuals (though only 116 were phenotyped). Mice were injected with *Listeria monocytogenes*; the phenotype is survival time (in hours). A large proportion of the mice (35/116) survived past the 240-hour time point and were considered to have recovered from the infection; their phenotype was recorded as 264.

A number of further example data sets will be used in this book. (For a summary of all data sets considered in the book, see Appendix C.) These have been compiled into an R package, R/qtlbook (known in R as the `qtlbook` package). It may be obtained from the website for the book (<http://www.rqtl.org/book>) and from the Comprehensive R Archive Network (CRAN, <http://cran.r-project.org>).

Additional example data may be obtained at the QTL Archive at The Jackson Laboratory (<http://cgd.jax.org/nav/qtllarchive1.htm> or use Google to search for “QTL Archive”). Most of the data sets are available in the “csv” format. One must register to access the data. As stated at the QTL Archive, “The authors of the datasets retain individual ownership of the data. We request, as a courtesy to the authors, that you alert them in advance of any publications that result from reanalysis of these data or obtain permission prior to redistribution of data or results.”

2.4 Data summaries

All of the data read by `read.cross` (including genotypes, phenotypes, and the genetic map) will be stored in a single object. (This object is stored in a quite complex form; see Sec. 2.6.1.) A number of functions are provided to get summary information about the object.

The most important function is `summary.cross`. In addition to providing a brief summary of the cross, it performs an extensive series of checks of the integrity of the data (for example, that there are the same number of individuals in the phenotype data as in the genotype data).

The data object for a QTL mapping experiment is assigned a “class” “`cross`”. R includes some simple object-oriented features, so that one may use the generic functions `summary` and `plot` on an object, and the relevant summary or plot is made.

For example, the following code loads the `listeria` data and displays a brief summary.

```
> data(listeria)
> summary(listeria)
```

F2 intercross	
No. individuals:	120
No. phenotypes:	2
Percent phenotyped:	96.7 100
No. chromosomes:	20
Autosomes:	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
X chr:	X
Total markers:	133
No. markers:	13 6 6 4 13 13 6 6 7 5 6 6 12 4 8 4 4 4 4 2
Percent genotyped:	88.5

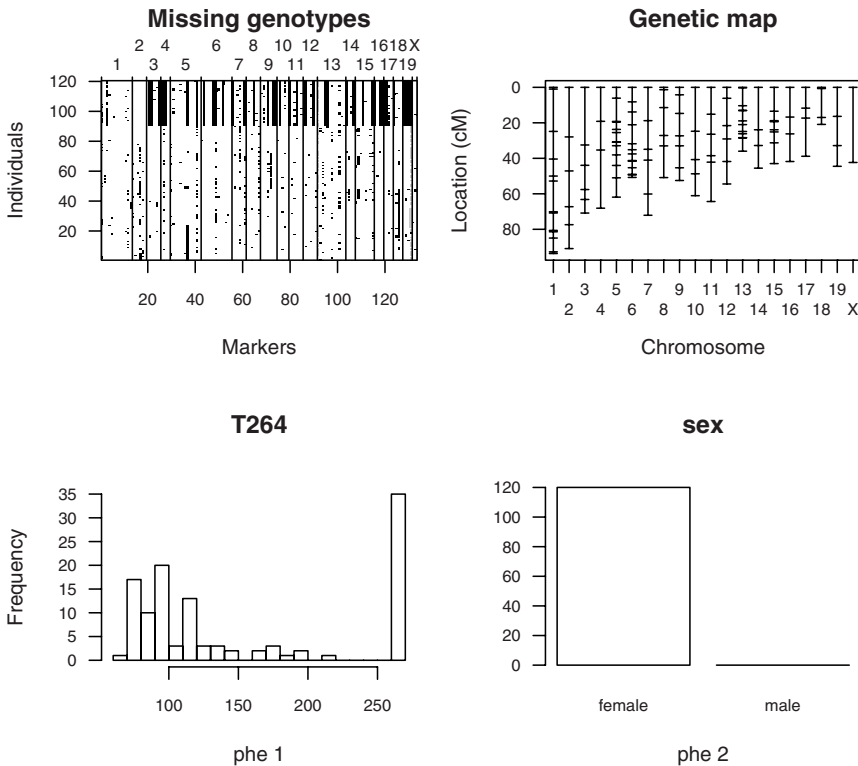


Figure 2.7. The summary plot of the `listeria` data provided by the `plot.cross` function, including the pattern of missing genotype data (upper left; black pixels indicate missing data), the genetic map of the typed markers (upper right), a histogram of the phenotype (lower left), and a bar plot of the sexes (lower right).

```
Genotypes (%):      CC:26.2  CB:48.9  BB:24  not BB:0
                   not CC:0.9
```

We see that this is an intercross with 120 individuals, that there are two phenotypes, and 20 chromosomes containing 133 markers, and with genotype completion of 88.5%.

In the above code, the generic `summary` function sees that `listeria` has class "cross" and passes it to the `summary.cross` function, which provides the actual summary.

Similarly, the following code provides a summary plot of the `listeria` data, and in this case the generic `plot` function passes `listeria` to the `plot.cross` function, which makes the plot (shown in Fig. 2.7).

```
> plot(listeria)
```

The individual panels in Fig. 2.7 may be obtained with the following code.

```

> plot.missing(listeria)
> plot.map(listeria)
> plot.pheno(listeria, 1)
> plot.pheno(listeria, 2)

```

The `plot.missing` function creates the plot with the pattern of missing genotype data. It takes an argument `reorder` which can be used to order the individuals according to their phenotype. The genetic map is obtained with `plot.map`. The function `plot.pheno` plots a phenotype, either as a histogram (using the R function `hist`) or as a bar plot (using the R function `barplot`), depending on the nature of the phenotype.

Finally, there are a variety of other functions for getting additional small pieces of information about a cross object. They are largely self-explanatory.

```

> nind(listeria)

[1] 120

> nphe(listeria)

[1] 2

> totmar(listeria)

[1] 133

> nchr(listeria)

[1] 20

> nmar(listeria)

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19  X
13  6  6  4 13 13  6  6  7  5  6  6 12  4  8  4  4  4  4  2

```

The function `nmar` gives the numbers of markers on individual chromosomes.

2.5 Simulating data

One can simulate QTL mapping data in R/qtl with the `sim.cross` function; it can simulate only additive QTL models. These basic facilities are described in the next subsection. More complex QTL models may also be simulated by making use of the QTL genotype data, which are stored in the object output by `sim.cross`. This will be described in the following subsection. Computer simulations are particularly useful for exploring the power to detect QTL and the precision of localization of QTL. For further details, see Sec. 6.6.

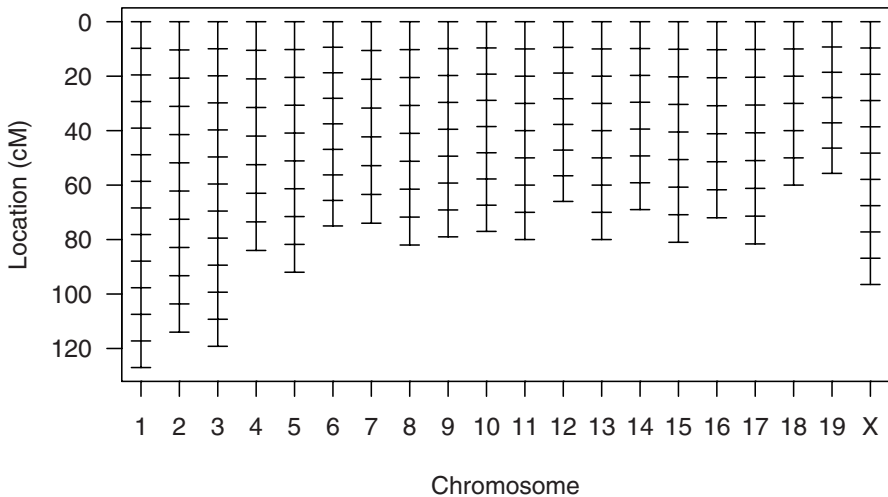


Figure 2.8. A genetic map, with approximately 10 cM marker spacing, modeled after the mouse genome and contained in the `map10` data set in R/qtl.

2.5.1 Additive models

The `sim.cross` function may be used to simulate a backcross or intercross with an additive QTL model. It requires, as input, a genetic map of markers. Such a map must be stored in a specific and rather complicated form (see Sec. 2.6.2), and so we first describe how to create such a map.

First, an example map, modeled after the mouse genome and having approximately evenly spaced markers (at ~ 10 cM) is provided with R/qtl in the data set `map10`. To access the object and plot the map, type the following.

```
> data(map10)
> plot(map10)
```

The plot is shown in Fig. 2.8. The marker spacing varies slightly across chromosomes so that the lengths of the chromosomes match those of the mouse genome.

Second, one may extract the genetic map from a QTL mapping data set with the `pull.map` function. For example, the following code extracts the map from the `listeria` data.

```
> data(listeria)
> listmap <- pull.map(listeria)
```

Finally, one may use `sim.map` to generate a map, with equally spaced markers or with markers placed randomly. Important arguments to `sim.map` include `len` (the cM lengths of the chromosomes), `n.markers` (the numbers of markers on the chromosomes), `anchor.tel` (indicates whether the ends of the chromosomes should be forced to have markers), `include.x` (whether the

final chromosome should be designated to be the X chromosome, versus all chromosomes being autosomes), and `eq.spacing` (whether markers should be spaced evenly).

For example, to create a map with a single autosome of length 200 cM and having markers equally spaced at 20 cM, type the following.

```
> mapA <- sim.map(200, 11, include.x=FALSE, eq.spacing=TRUE)
```

To create a map with 19 autosomes and an X chromosome, chromosomes all of length 100 cM, and each containing 10 randomly positioned markers, though ensuring one marker at each end of each chromosome, we would type the following.

```
> mapB <- sim.map(rep(100, 20), 10)
```

A similar map, but without anchoring the telomeres, would be obtained as follows.

```
> mapC <- sim.map(rep(100, 20), 10, anchor.tel=FALSE)
```

Finally, to get a map with four autosomes of lengths 50, 75, 100, and 125 cM, respectively, and with equally spaced markers at a 5 cM spacing, type the following.

```
> L <- c(50, 75, 100, 125)
```

```
> mapD <- sim.map(L, L/5+1, eq.spacing=TRUE, include.x=FALSE)
```

Note that one can use the `summary.map` function to get a short summary of a genetic map; it works much like the `summary.cross` function described in Sec. 2.4. We can get a summary of the `mapD` object, created above, as follows.

```
> summary(mapD)
```

	n.mar	length	ave.spacing	max.spacing
1	11	50	5	5
2	16	75	5	5
3	21	100	5	5
4	26	125	5	5
overall	74	350	5	5

With a genetic map in hand, we can now turn to the simulation of the actual data. The following code simulates data for a backcross of 100 individuals, with complete and error-free genotype data, and markers placed according to the genetic map in `map10`.

```
> simA <- sim.cross(map10, n.ind=100, type="bc")
```

We would simulate an intercross in the same way, using `type="f2"`.

If QTL are to be simulated, we must specify the model via the `model` argument, which should be a matrix with three columns for a backcross and four columns for an intercross. The first column in the matrix gives the chromosomes on which the QTL sit and the second column gives their cM positions.

The third column contains the additive effect of each QTL: in a backcross, the difference between the phenotype averages in heterozygotes and homozygotes; in an intercross, half the difference between phenotype averages for the homozygotes. In an intercross, there must be a fourth column giving the dominance effect for each QTL (the difference between the average phenotype for the heterozygotes and the midpoint between the average phenotypes for the homozygotes).

Phenotypes are simulated from a normal distribution with residual variance $\sigma^2 = 1$. Thus, in a backcross, if there is one QTL with additive effect a , the proportion of the phenotypic variance explained by the QTL (i.e., the heritability due to the QTL) will be $a^2/4/(a^2/4 + 1)$. In an intercross with one QTL exhibiting no dominance, the proportion of the phenotypic variance explained is $a^2/2/(a^2/2 + 1)$.

Let us first simulate a backcross with two additive QTL, each responsible for 8% of the phenotypic variance. Place the first at 50 cM on chromosome 1 and the second at 65 cM on chromosome 14. We must first find the additive effects that correspond to 8% phenotypic variance. Since the QTL are unlinked and have the same size effect, we need $(a^2/4)/[2(a^2/4) + 1] = 0.08$. Solving for a , we obtain $a = \sqrt{4 \times 0.08/(1 - 2 \times 0.08)}$.

```
> a <- 2 * sqrt(0.08 / (1 - 2 * 0.08))
> mymodel <- rbind(c(1, 50, a), c(14, 65, a))
> simB <- sim.cross(map10, type="bc", n.ind=200, model=mymodel)
```

We use the `c` function to combine the chromosome, position and effect of each QTL into a vector, and then `rbind` to combine the two into a matrix (`rbind` makes them rows in the matrix).

As a further example, we simulate an intercross of 250 individuals with three QTL, two having no dominance but with effects in the opposite directions and a third being strictly dominant. Let's have the first two QTL be linked on chromosome 3 at positions 40 cM and 65 cM, and place the third on chromosome 4 at 5 cM. For simplicity, let's set the effects at 0.5.

```
> mymodel2 <- rbind(c(3, 40, 0.5, 0), c(3, 65, -0.5, 0),
+                  c(4, 5, 0.5, 0.5))
> simC <- sim.cross(map10, type="f2", n.ind=250, model=mymodel2)
```

By default, there are no errors in the genotype data. Errors can be included at random via the `error.prob` argument. Genotype data are also, by default, complete. The genotype data can be missing at random with some probability via the `missing.prob` argument. And so we can repeat our backcross simulation with 1% genotyping errors and 5% missing data as follows.

```
> simD <- sim.cross(map10, type="bc", n.ind=200, model=mymodel,
+                  error.prob=0.01, missing.prob=0.05)
```

Random missing genotype data is rather artificial. For more realistic missing data, we can simulate an intercross of the same size as the `listeria` data

and apply the missing data observed in that data set. This is not so simple, due to the complexity of the cross data objects and the need for a loop over chromosomes, and so the following code has little chance of being understood by the novice.

```
> data(listeria)
> listmap <- pull.map(listeria)
> simE <- sim.cross(listmap, type="f2", n.ind=nind(listeria),
+                   model=mymodel2)
> for(i in 1:nchr(simE))
+   simE$geno[[i]]$data[ is.na(listeria$geno[[i]]$data) ] <- NA
```

By default, simulations are performed assuming no crossover interference at meiosis. One may also simulate the crosses under the χ^2 model or the Stahl model. (See Sec. 2.7 for references.) The χ^2 model has a single parameter, m , which is a non-negative integer; $m = 0$ corresponds to no interference. With $m > 0$, it is assumed that, on the four-strand bundle at meiosis, chiasmata and intermediate points are thrown down at random (according to a Poisson process), and that every $(m + 1)$ st point is a chiasma. No chromatid interference is assumed, so that the particular strands involved in each chiasma are at random, independent between chiasmata. As a result, the crossovers on a random meiotic product may be obtained by “thinning” the chiasmata independently with probability $1/2$. (That is, each chiasma has $1/2$ chance of being a crossover on the random product, with independence between chiasmata.) In the Stahl model, chiasmata arise according to two independent mechanisms, one following a χ^2 model and the other exhibiting no interference; the observed chiasma locations are the superposition of the two processes. There is one additional parameter, p , giving the proportion of chiasmata to come from the mechanism exhibiting no interference.

We can simulate under the χ^2 model and the Stahl model via the arguments `m` and `p` to `sim.cross`. By default, `m=0` (in which case `p` is irrelevant), indicating no crossover interference. The mouse exhibits strong crossover interference with $m \approx 10$. We can repeat our previous simulation, but with recombination according to a $\chi^2(m = 10)$ model as follows.

```
> simF <- sim.cross(map10, type="f2", n.ind=250, model=mymodel2,
+                   m=10)
```

We can simulate from the Stahl model, with $m = 10$ and $p = 0.1$, as follows.

```
> simG <- sim.cross(map10, type="f2", n.ind=250, model=mymodel2,
+                   m=10, p=0.1)
```

2.5.2 More complex models

The simulations in the previous section were restricted to strictly additive QTL models and with residual variation following a normal distribution with

variance $\sigma^2 = 1$. However, the QTL genotype data are stored as a matrix within the output of `sim.cross`; with these data one may simulate data from essentially any QTL model.

First, let us simulate two QTL exhibiting epistasis. Consider a backcross of 200 individuals, with a QTL located at 25 cM on chromosome 4 and another at 45 cM on chromosome 5. Assume that an effect is seen only if an individual is homozygous at both QTL, in which case the phenotype is reduced by one unit.

We begin by simulating QTL having no effect, just so that their genotypes may be obtained, but so that the simulated phenotype will follow a normal(0,1) distribution, independent of genotype. We then modify the phenotype for individuals who are homozygous at both QTL. This requires a bit of mucking about in the cross data object.

```
> data(map10)
> nullmodel <- rbind(c(4, 25, 0), c(5, 45, 0))
> episim <- sim.cross(map10, type="bc", n.ind=200,
+                      model=nullmodel)
> qtlg <- episim$qtlgeno
> wh <- qtlg[,1]==1 & qtlg[,2]==1
> episim$pheno[wh, 1] <- episim$pheno[wh, 1] - 1
```

In the fifth line, we pull out the QTL genotype data. (The columns are the QTL; the rows are the individuals.) In the sixth line, we identify the individuals that are homozygous at both QTL. (Internally, in a backcross, 1 and 2 correspond to the homozygous and heterozygous genotypes, respectively. In an intercross, 1 and 3 are the two homozygous genotypes and 2 is the heterozygous genotype.)

We might create a binary version of this phenotype by thresholding at 1. (Individuals with quantitative phenotype > 1 become affected; the others are unaffected.) We can paste this into the simulated data as a second phenotype.

```
> binphe <- as.numeric(episim$pheno[,1] > 1)
> episim$pheno$affected <- binphe
```

There will now be a second phenotype named “affected” with 1 and 0 indicating affected and unaffected, respectively.

Finally, we might assign sexes to the individuals at random, and include a sex difference in the phenotype and even a difference in the effect of the QTL in the two sexes (a QTL $\times$ sex interaction). We’ll create a third phenotype with these features, and place “sex” in the data as a fourth phenotype. Here, 0 and 1 correspond to females and males, respectively.

```
> sex <- sample(0:1, nind(episim), replace=TRUE)
> phe3 <- rnorm(nind(episim), 0, 1)
> phe3[wh & sex==0] <- phe3[wh & sex==0] - 1.5
> phe3[wh & sex==1] <- phe3[wh & sex==1] - 0.5
```

```
> episim$pheno$pheno3 <- phe3
> episim$pheno$sex <- sex
```

We use the R function `sample` to sample with replacement from the vector (0, 1), and `rnorm` to simulate standard normal data. The epistasis pattern for the two QTL is as before, but the effects are different in the two sexes. We reuse the `wh` object, created above, that indicated the individuals who were homozygous at both QTL.

2.6 Internal data structure

In this section, we describe the internal data structures used by R/qtl for cross and genetic map objects and the R syntax required to get access to the data. Other data structures (such as those produced by the `scanone` and `scantwo` functions) will be described in later chapters. This section is quite technical and will require a reasonably detailed understanding of R, and so it should probably be skipped initially. The choice of data structures required some balance between ease of programming and simplicity for the user interface. The syntax for references to certain pieces of the internal data can be quite complicated.

2.6.1 Experimental cross

We describe the internal data structure used by R/qtl for QTL mapping data; we will look at the data set `hyper` as an example. First, the object has a “class,” which indicates that it corresponds to data for an experimental cross, and gives the cross type. By having class “`cross`”, the functions `plot` and `summary` know to send the data to `plot.cross` and `summary.cross`.

```
> data(hyper)
> class(hyper)

[1] "bc"      "cross"
```

As you can see, the class is a two-element vector containing first a character string indicating the cross type (“`bc`” or “`f2`”) and second “`cross`” to indicate that it is an experimental cross.

Every cross object is a list with two components, one containing the genotype data and genetic maps and the other containing the phenotype data.

```
> names(hyper)

[1] "geno"  "pheno"
```

The phenotype data is simply a matrix (more strictly a data frame) with rows corresponding to individuals and columns corresponding to phenotypes. We look at the phenotypes for the first five individuals as follows.

```
> hyper$pheno[1:5,]
```

```
      bp  sex
1 109.6 male
2 109.8 male
3 110.1 male
4 110.6 male
5 115.0 male
```

The first phenotype is the blood pressure of each mouse; the second phenotype indicates their sex. (In this case, all mice are male.) The phenotypes can be either numeric or factors. The sex phenotype can be coded 0/1, f/m, F/M, or female/male for female/male; in all but the first case, it must be a factor.

The genotype data is a list with components corresponding to chromosomes. Each chromosome has a name and a class. The class for a chromosome is "A" or "X", for autosomes or the X chromosome, respectively.

```
> names(hyper$geno)
```

```
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11"
[12] "12" "13" "14" "15" "16" "17" "18" "19" "X"
```

```
> sapply(hyper$geno, class)
```

```
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
"A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
16 17 18 19  X
"A" "A" "A" "A" "X"
```

Each component of `geno` is itself a list with two components, `data` (containing the marker genotype data) and `map` (containing the positions of the markers, in cM). The genotype data are coded 1/2 for homozygotes and heterozygotes in a backcross, and 1/2/3/4/5 for the genotypes AA/AB/BB/not BB/not AA in an intercross.

```
> names(hyper$geno[[3]])
```

```
[1] "data" "map"
```

```
> hyper$geno[[3]]$data[91:94,]
```

	D3Mit164	D3Mit6	D3Mit11	D3Mit14	D3Mit44	D3Mit19
91	2	1	1	1	1	1
92	1	1	1	1	1	1
93	NA	2	NA	NA	NA	NA
94	NA	2	NA	NA	NA	NA

```
> hyper$geno[[3]]$map
```

D3Mit164	D3Mit6	D3Mit11	D3Mit14	D3Mit44	D3Mit19
2.2	17.5	37.2	44.8	57.9	66.7

On the X chromosome, all individuals are coded with genotypes 1/2. We use the phenotypes `sex` and `pgm`, if they are available, to recode these as AA/AB/BB/AY/BY before later analysis. The 1/2 codes simplify the use of the HMM algorithms (as in `calc.genoprob`, to calculate genotype probabilities), as all individuals may be treated as a backcross.

That completes the description of the raw data. However, other information may exist in a cross object, as when one runs `calc.genoprob`, `sim.geno`, or `calc.errorlod`, the output is the input cross object with the derived data attached to each component (the chromosomes) of the `geno` component.

```
> names(hyper$geno[[3]])

[1] "data" "map"

> hyper <- calc.genoprob(hyper, step=10, error.prob=0.01)
> names(hyper$geno[[3]])

[1] "data" "map" "prob"

> hyper <- sim.geno(hyper, step=10, n.draws=2, error.prob=0.01)
> names(hyper$geno[[3]])

[1] "data" "map" "prob" "draws"

> hyper <- calc.errorlod(hyper, error.prob=0.01)
> names(hyper$geno[[3]])

[1] "data" "map" "prob" "draws" "errorlod"
```

The structure of the individual components that were added is relatively self-explanatory.

Finally, when one runs `est.rf`, a matrix containing the pairwise recombination fractions and LOD scores is added to the cross object.

```
> names(hyper)

[1] "geno" "pheno"

> hyper <- est.rf(hyper)
> names(hyper)

[1] "geno" "pheno" "rf"
```

The `hyper$rf` object is a matrix. Values on the diagonal are the number of individuals that were genotyped for the corresponding marker. Values above the diagonal are LOD scores for a test of linkage; values below the diagonal are estimated recombination fractions.

```
> hyper$rf[1:4,1:4]
```

	D1Mit296	D1Mit123	D1Mit156	D1Mit178
D1Mit296	92.0000	11.4201	3.1422	0.6321
D1Mit123	0.1413	92.0000	9.9274	0.6321
D1Mit156	0.3043	0.1630	250.0000	2.9045
D1Mit178	0.1667	0.1667	0.2449	49.0000

The function `clean.cross` may be used to remove the intermediate results from a cross object (such as those created with `calc.genoprob` and `est.rf`), as follows.

```
> hyper <- clean(hyper)
> names(hyper)

[1] "geno" "pheno"

> names(hyper$geno[[3]])

[1] "data" "map"
```

2.6.2 Genetic map

A genetic map object, as produced by `sim.map` or as extracted from a cross object with `pull.map`, also has a somewhat complex form. We will look at the data set `map10`, a genetic map modeled after the mouse genome. Such a map object has class `"map"` so that `plot` and `summary` will call `plot.map` and `summary.map`, respectively.

```
> data(map10)
> class(map10)

[1] "map"
```

The map is a list whose components are the individual chromosomes. Each chromosome has class either `"A"` or `"X"` according to whether it is an autosome or the X chromosome.

```
> names(map10)

[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11"
[12] "12" "13" "14" "15" "16" "17" "18" "19" "X"

> sapply(map10, class)

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
"A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
16 17 18 19  X
"A" "A" "A" "A" "X"
```

The individual chromosomes are vectors specifying the marker locations in cM, with names being the marker names.

```
> map10[[15]]
D15M1 D15M2 D15M3 D15M4 D15M5 D15M6 D15M7 D15M8 D15M9
  0.00 10.12 20.25 30.38 40.50 50.62 60.75 70.88 81.00
attr(,"class")
[1] "A"
```

2.7 Further reading

Broman and Heath (2007) discuss the management and manipulation of genetic data. They emphasize the need for biologists to learn to program, and the value of the Perl programming language for geneticists. While they focus on human linkage data, the general principles apply to all genetic data.

Useful Perl books include *Learning Perl* (Schwartz *et al.*, 2008) for beginners, *Programming Perl* (Wall *et al.*, 2000) as a reference, and *Perl Cookbook* (Christiansen and Torkington, 2003) for its recipes encompassing many common tasks. These books, plus a couple of others, may be purchased together on a CD for a very good price: the *Perl CD Bookshelf*, available from O'Reilly Media.

Regarding the χ^2 model for crossover interference, see Zhao *et al.* (1995). The Stahl model was described in Copenhaver *et al.* (2002).