A brief tour of R/qtl

Karl W Broman

Department of Biostatistics, Johns Hopkins University

http://www.rqtl.org

27 October 2006

Overview of R/qtl

R/qtl is an extensible, interactive environment for mapping quantitative trait loci (QTL) in experimental crosses. It is implemented as an add-on package for the freely available and widely used statistical language/software R (see www.r-project.org). The development of this software as an add-on to R allows us to take advantage of the basic mathematical and statistical functions, and powerful graphics capabilities, that are provided with R. Further, the user will benefit by the seamless integration of the QTL mapping software into a general statistical analysis program. Our goal is to make complex QTL mapping methods widely accessible and allow users to focus on modeling rather than computing.

A key component of computational methods for QTL mapping is the hidden Markov model (HMM) technology for dealing with missing genotype data. We have implemented the main HMM algorithms, with allowance for the presence of genotyping errors, for backcrosses, intercrosses, and phase-known four-way crosses.

The current version of R/qtl includes facilities for estimating genetic maps, identifying genotyping errors, and performing single-QTL genome scans and two-QTL, two-dimensional genome scans, by interval mapping (with the EM algorithm), Haley-Knott regression, and multiple imputation. All of this may be done in the presence of covariates (such as sex, age or treatment). One may also fit higher-order QTL models by multiple imputation.

R/qtl is distributed as source code for Unix or compiled code for Windows or Mac OS X. R/qtl is released under the GNU General Public License. To download the software, you must agree to the terms in that license.

Overview of R

R is an open-source implementation of the S language. As described on the R-project homepage (www.r-project.org):

> R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.
>
> The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. Most of the user-visible functions in R are written in R. It is possible for the user to interface to procedures written in the C, C++, or FORTRAN languages for efficiency. The R distribution contains functionality for a large number of statistical procedures. Among these are: linear and generalized linear models, nonlinear regression models, time series analysis, classical parametric and nonparametric tests, clustering and smoothing. There is also a large set of functions which provide a flexible graphical environment for creating various kinds of data presentations. Additional modules are available for a variety of specific purposes.

R is freely available for Windows, Unix and Mac OS X, and may be downloaded from the Comprehensive R Archive Network (CRAN; cran.r-project.org).

Learning R may require a formidable investment of time, but it will definitely be worth the effort. Numerous free documents on getting started with R are available on CRAN. In additional, several books are available. The most important book on R is Venables and Ripley (2002) *Modern Applied Statistics with S*, 4th edition. Dalgaard (2002) *Introductory Statistics with R* provides a more gentle introduction.

Citation for R/qtl

To cite R/qtl in publications, use

> Broman KW, Wu H, Sen S, Churchill GA (2003) R/qtl: QTL mapping in experimental crosses. Bioinformatics 19:889-890

Selected R/qtl functions

Sample data	badorder	An intercross with misplaced markers
	bristle3	Data on bristle number for Drosophila chromosome 3
	bristleX	Data on bristle number for Drosophila X chromosome
	fake.4way	Simulated data for a 4-way cross
	fake.bc	Simulated data for a backcross
	fake.f2	Simulated data for an F_2 intercross
	hyper	Backcross data on salt-induced hypertension
	listeria	Intercross data on Listeria monocytogenes susceptibility
	map10	A genetic map modeled after the mouse genome (10 cM spacing)
Input/output	read.cross	Read data for a QTL experiment
	write.cross	Write data for a QTL experiment to a file
Simulation	sim.cross	Simulate a QTL experiment
	sim.map	Generate a genetic map
Summaries	geno.table	Create table of genotype distributions
	plot.cross	Plot various features of a cross object
	plot.missing	Plot grid of missing genotypes
	plot.pheno	Histogram or bar plot of a phenotype
	plot.info	Plot the proportion of missing genotype data
	summary.cross	Print summary of QTL experiment
	summary.map	Print summary of a genetic map
	nchr, nind, nmar, nphe, totmar, nmissing	
Data manipulation	clean.cross	Remove intermediate calculations from a cross
	drop.markers	Remove a list of markers
	drop.nullmarkers	Remove markers without data
	fill.geno	Fill in holes in genotype data by imputation or Viterbi
	pull.map	Pull out the genetic map from a cross
	replace.map	Replace the genetic map of a cross
	subset.cross	Select a subset of chromosomes and/or individuals from a cross
	switch.order	Switch the order of markers on a chromosome
	movemarker	Move a marker from one chromosome to another
HMM engine	argmax.geno	Reconstruct underlying genotypes by the Viterbi algorithm
	calc.genoprob	Calculate conditional genotype probabilities
	sim.geno	Simulate genotypes given observed marker data
QTL mapping	scanone	Genome scan with a single QTL model
	scantwo	Two-dimensional genome scan with a two-QTL model
	lodint	Calculate a LOD support interval
	bayesint	Calculate an approximate Bayes credible interval
	plot.scanone	Plot output for a one-dimensional genome scan
	plot.scantwo	Plot output for a two-dimensional genome scan
	summary.scanone	Print summary of scanone output
	summary.scantwo	Print summary of scantwo output
	effectplot	Plot phenotype means of genotype groups defined by 1 or 2 markers
	plot.pxg	Like effectplot, but as a dot plot of the phenotypes
Genetic mapping	est.map	Estimate genetic map
	est.rf	Estimate pairwise recombination fractions
	plot.map	Plot genetic map(s)
	plot.rf	Plot recombination fractions
	ripple	Assess marker order by permuting groups of adjacent markers
	summary.ripple	Print summary of ripple output
Genotyping errors	calc.errorlod	Calculate Lincoln & Lander (1992) error LOD scores
	top.errorlod	List genotypes with highest error LOD values
	plot.geno	Plot observed genotypes, flagging likely errors
Multiple QTL models	makeqtl	Make a qtl object for use by fitqtl
	fitqtl	Fit a multiple QTL model, using multiple imputation
	summary.fitqtl	Get summary of the result of fitqtl
	scanqtl	Perform a multi-dimensional genome scan, using multiple imputation

Preliminaries

Use of the R/qtl package requires considerable knowledge of the R language/environment. We hope that the examples presented here will be understandable with little prior knowledge of R, especially because we neglect to explain the syntax of R. Several books, as well as some free documents, are available to assist the user in learning R; see the R project website cited above. We assume here that the user is running either Windows or Mac OS X.

1. To start R, double-click its icon.
2. To exit, type:

   ```
   q()
   ```

   Click yes or no to save or discard your work.
3. Load the R/qtl package:

   ```
   library(qtl)
   ```
4. View the objects in your workspace:

   ```
   ls()
   ```
5. The best way to get help on the functions and data sets in R (and in R/qtl) is via the html version of the help files. One way to get access to this is to type

   ```
   help.start()
   ```

   This should open a browser with the main help menu. If you then click on Packages → qtl, you can see all of the available functions and datasets in R/qtl. For example, look at the help file for the function `read.cross`.

   An alternative method to view this help file is to type one of the following:

   ```
   help(read.cross)
   ?read.cross
   ```

   The html version of the help files are somewhat easier to read, and allow use of hotlinks between different functions.
6. All of the code in this tutorial is available as a file from which you may copy and paste into R, if you prefer that to typing. Type the following within R to get access to the file:

   ```
   url.show("http://www.rqtl.org/rqtltour.R")
   ```

Data import

A difficult first step in the use of most data analysis software is the import of data. With R/qtl, one may import data in several different formats by use of the function `read.cross`. The internal data structure used by R/qtl is rather complicated, and is described in the help file for `read.cross`. We won't discuss data import any further here, except to say that the comma-delimited format (`"csv"`) is recommended. If you have trouble importing data, send an email to Karl Broman (`kbroman@jhsph.edu`) perhaps attaching examples of your data files. (Such data will be kept confidential.)

Example 1: Hypertension

As a first example, we consider data from an experiment on hypertension in the mouse (Sugiyama et al., Genomics 71:70-77, 2001), kindly provided by Bev Paigen and Gary Churchill.

1. First, get access to the data, see that it is in your workspace, and view its help file. These data are included with the R/qtl package, and so you can get access to the data with the function `data()`.

   ```
   data(hyper)
   ls()
   ?hyper
   ```
2. We will postpone discussion of the internal data structure used by R/qtl until later. For now we'll just say that the data `hyper` has "class" `"cross"`. The function `summary.cross` prints summary information on such data. We can call that function directly, or we may simply use `summary` and the data is sent to the appropriate function according to its class.

   ```
   summary(hyper)
   ```

Several other utility functions are available for getting summary information on the data. Hopefully these are self-explanatory.

```
nind(hyper)
nphe(hyper)
nchr(hyper)
totmar(hyper)
nmar(hyper)
```

3. Plot a summary of these data.

   ```
   plot(hyper)
   ```

   In the upper left, black pixels indicate missing genotype data. Note that one marker has no genotype data. In the upper right, the genetic map of the markers is shown. In the lower left, a histogram of the phenotype is shown.

   The Windows version of R has a slick method for recording graphs, so that one may page up and down through a series of plots. To initiate this, click (on the menu bar) History → Recording.

   We may plot the individual components of the above multi-plot figure as follows.

   ```
   plot.missing(hyper)
   plot.map(hyper)
   plot.pheno(hyper, pheno.col=1)
   ```

   We can plot the genetic map with marker names, but they can be rather difficult to read. The following code plots the map with marker names for chromosomes 1, 4, 6, 7 and 15.

   ```
   plot.map(hyper, chr=c(1, 4, 6, 7, 15), show.marker.names=TRUE)
   ```

4. Note the odd pattern of missing data; we may make this missing data plot with the individuals ordered according to the value of their phenotype.

   ```
   plot.missing(hyper, reorder=TRUE)
   ```

   We see that, for most markers, only individuals with extreme phenotypes were genotyped. At many markers (in regions of interest), markers were typed only on recombinant individuals.

5. The function `drop.nullmarkers` may be used to remove markers that have no genotype data (such as the marker on chromosome 14). A call to `totmar` will show that there are now 173 markers (rather than 174, as there were initially).

   ```
   hyper <- drop.nullmarkers(hyper)
   totmar(hyper)
   ```

6. Estimate recombination fractions between all pairs of markers, and plot them. This also calculates LOD scores for the test of H_0: $r = 1/2$. The plot of the recombination fractions can be either with recombination fractions in the upper part and LOD scores below, or with just recombination fractions or just LOD scores. Note that red corresponds to a small recombination fraction or a big LOD score, while blue is the reverse. Gray indicates missing values.

   ```
   hyper <- est.rf(hyper)
   plot.rf(hyper)
   plot.rf(hyper, chr=c(1,4))
   ```

   There are some very strange patterns in the recombination fractions, but this is due to the fact that some markers were typed largely on recombinant individuals.

   For example, on chr 6, the tenth marker shows a high recombination fraction with all other markers on the chromosome, but a plot of the missing data shows that this marker was typed only on a selected number of individuals (largely those showing recombination events across the interval).

   ```
   plot.rf(hyper, chr=6)
   plot.missing(hyper, chr=6)
   ```

7. Re-estimate the genetic map (keeping the order of markers fixed), and plot the original map against the newly estimated one.

   ```
   newmap <- est.map(hyper, error.prob=0.01)
   plot.map(hyper, newmap)
   ```

We see some map expansion, especially on chromosomes 6, 13 and 18. It is questionable whether we should replace the map or not. Keep in mind that the previous map locations are based on a limited number of meioses. If one wished to replace the genetic map with the estimated one, it could be done as follows:

```
hyper <- replace.map(hyper, newmap)
```

This replaces the map in the `hyper` data with `newmap`.

8. We now turn to the identification of genotyping errors. In the following, we calculate the error LOD scores of Lincoln and Lander (1992). A LOD score is calculated for each individual at each marker; large scores indicate likely genotyping errors.

   ```
   hyper <- calc.errorlod(hyper, error.prob=0.01)
   ```

   This calculates the genotype error LOD scores and inserts them into the `hyper` object.

   The function `top.errorlod` gives a list of genotypes that may be in error. Error LOD scores < 4 can probably be ignored.

   ```
   top.errorlod(hyper)
   ```

   Note that the results will be different, depending on whether you used `replace.map` above. If you did, you will get an indication of potential errors on chromosome 16. If you didn't, you will get an indication of potential errors on chromosomes 1, 11 and 17.

9. The function `plot.geno` may be used to inspect the observed genotypes for a chromosome, with likely genotyping errors flagged. Of course, it's difficult to look at too many individuals at once. Note that white = AA and black = AB (for a backcross).

   ```
   plot.geno(hyper, chr=16, ind=c(24:34, 71:81))
   ```

   We don't have any utilities for fixing any apparent errors; it would be best to go back to the raw data.

10. The function `plot.info` plots a measure of the proportion of missing genotype information in the genotype data. The missing information is calculated in two ways: as entropy, or via the variance of the conditional genotypes, given the observed marker data. (See the help file, using `?plot.info`.)

    ```
    plot.info(hyper)
    plot.info(hyper, chr=c(1,4,15))
    plot.info(hyper, chr=c(1,4,15), method="entropy")
    plot.info(hyper, chr=c(1,4,15), method="variance")
    ```

11. We now, finally, get to QTL mapping.

    The core of R/qtl is a set of functions which make use of the hidden Markov model (HMM) technology to calculate QTL genotype probabilities, to simulate from the joint genotype distribution and to calculate the most likely sequence of underlying genotypes (all conditional on the observed marker data). This is done in a quite general way, with possible allowance for the presence of genotyping errors. Of course, for convenience we assume no crossover interference.

    The function `calc.genoprob` performs calculates QTL genotype probabilities. These are needed for most of the QTL mapping functions. The argument `step` indicates the step size (in cM) at which the probabilities are calculated, and determines the step size at which later LOD scores are calculated.

    ```
    hyper <- calc.genoprob(hyper, step=1, error.prob=0.01)
    ```

    We may now use the function `scanone` to perform a single-QTL genome scan with a normal model. We may use maximum likelihood via the EM algorithm (Lander and Botstein 1989) or use Haley-Knott regression (Haley and Knott 1992).

    ```
    out.em <- scanone(hyper)
    out.hk <- scanone(hyper, method="hk")
    ```

    We may also use the multiple imputation method of Sen and Churchill (2001). This requires that we first use `sim.geno` to simulate from the joint genotype distribution, given the observed marker data. Again, the argument `step` indicates the step size at which the imputations are performed and determines the step size at which LOD scores will be calculated. The `n.draws` indicates the number of imputations to perform. Larger values give more precise results but require considerably more computer memory and computation time.

    ```
    hyper <- sim.geno(hyper, step=2, n.draws=16, error.prob=0.01)
    out.imp <- scanone(hyper, method="imp")
    ```

12. The output of scanone has class `"scanone"`; the function `summary.scanone` displays the maximum LOD score on each chromosome for which the LOD exceeds a specified threshold.

```
summary(out.em)
summary(out.em, threshold=3)
summary(out.hk, threshold=3)
summary(out.imp, threshold=3)
```

13. The function `max.scanone` returns just the highest peak from output of `scanone`.

```
max(out.em)
max(out.hk)
max(out.imp)
```

14. We may also plot the results. `plot.scanone` can plot up to three genome scans at once, provided that they conform appropriately. Alternatively, one may use the argument `add`.

```
plot(out.em, chr=c(1,4,15))
plot(out.em, out.hk, out.imp, chr=c(1,4,15))
plot(out.em, chr=c(1,4,15))
plot(out.hk, chr=c(1,4,15), col="blue", add=TRUE)
plot(out.imp, chr=c(1,4,15), col="red", add=TRUE)
```

15. The function `scanone` may also be used to perform a permutation test to get a genome-wide LOD significance threshold. For Haley-Knott regression, this can be quite fast.

```
operm.hk <- scanone(hyper, method="hk", n.perm=1000)
```

The permutation output has class `"scanoneperm"`. The function `summary.scanoneperm` can be used to get significance thresholds.

```
summary(operm.hk, alpha=0.05)
```

In addition, if the permutations results are included in a call to `summary.scanone`, you can estimated genome-scan-adjusted p-values for inferred QTL, and can get a report of all chromosomes meeting a certain significance level, with the corresponding LOD threshold calculated automatically.

```
summary(out.hk, perms=operm.hk, alpha=0.05, pvalues=TRUE)
```

16. We should mention at this point that the function `save.image` may be used to save your workspace to disk. If R crashes, you will wish you had used this.

```
save.image()
```

17. The function `scantwo` performs a two-dimensional genome scan with a two-QTL model. For every pair of positions, it calculates a LOD score for the full model (two QTL plus interaction) and a LOD score for the additive model (two QTL but no interaction). This be quite time consuming, and so you may wish to do the calculations on a coarser grid.

```
hyper <- calc.genoprob(hyper, step=5, error.prob=0.01)
out2.hk <- scantwo(hyper, method="hk")
```

One can also use `method="em"` or `method="imp"`, but they are even more time consuming.

18. The output of `scantwo` has class `"scantwo"`; there are functions for obtaining summaries and plots, of course.

The summary function considers each pair of chromosomes, and calculates the maximum LOD score for the full model (M_f) and the maximum LOD score for the additive model (M_a). These two models are allowed to be maximized at different positions. We futher calculate a LOD score for a test of epistasis, $M_i = M_f - M_a$, and two LOD scores that concern evidence for a second QTL: M_{fv1} is the LOD score comparing the full model to the best single-QTL model and M_{av1} is the LOD score comparing the additive model to the best single-QTL model.

In the summary, we must provide five thresholds, for M_f, M_{fv1}, M_i, M_a, and M_{av1}, respectively. Call these T_f, T_{fv1}, T_i, T_a, and T_{av1}. We then report those pairs of chromosomes such that at least one of the following holds:

- $M_f \geq T_f$ and ($M_{fv1} \geq T_{fv1}$ or $M_i \geq T_i$)
- $M_a \geq T_a$ and $M_{av1} \geq T_{av1}$

The thresholds can be obtained by a permutation test (see below), but this is extremely time-consuming. For a mouse backcross, we suggest the thresholds (6.0, 4.7, 4.4, 4.7, 2.6) for the full, conditional-interactive, interaction, additive, and conditional-additive LOD scores, respectively. For a mouse intercross, we suggest the thresholds (9.1, 7.1, 6.3, 6.3, 3.3) for the full, conditional-interactive, interaction, additive, and conditional-additive LOD scores, respectively. These were obtained by 10,000 simulations of crosses with 250 individuals, markers at a 10 cM spacing, and analysis by Haley-Knott regression.

```
summary(out2.hk, thresholds=c(6.0, 4.7, 4.4, 4.7, 2.6))
```

The appropriate decision rule is not yet completely clear. I am inclined to ignore M_i and to choose genome-wide thresholds for the other four based on a permutation, using a common significance level for all four. M_i would be ignored if we gave it a very large threshold, as follows.

```
summary(out2.hk, thresholds=c(6.0, 4.7, Inf, 4.7, 2.6))
```

19. Plots of `scantwo` results are created via `plot.scantwo`.

```
plot(out2.hk)
plot(out2.hk, chr=c(1,4,6,15))
```

By default, the upper-left triangle contains epistasis LOD scores and the lower-right triangle contains the LOD scores for the full model. The color scale on the right indicates separate scales for the epistasis and joint LOD scores (on the left and right, respectively).

20. The function `max.scantwo` returns the two-locus positions with the maximum LOD score for the full and additive models.

```
max(out2.hk)
```

21. One may also use `scantwo` to perform permutation tests in order to obtain genome-wide LOD significance thresholds. These can be extremely time consuming, though with the Haley-Knott regression and multiple imputation methods, there is a trick that may be used in some caes to dramatically speed things up. So we'll try 100 permutations by the Haley-Knott regression method and hope that your computer is sufficiently fast.

```
operm2.hk <- scantwo(hyper, method="hk", n.perm=100)
```

We can gain use `summary` to get LOD thresholds.

```
summary(operm2.hk)
```

And again these may be used in the summary of the `scantwo` output to calculate thresholds and p-values. If you want to ignore the LOD score for the interaction in the rule about what chromosome pairs to report, give $\alpha = 0$, corresponding to a threshold $T = \infty$.

```
summary(out2.hk, perms=operm2.hk, pvalues=TRUE,
        alphas=c(0.05, 0.05, 0, 0.05, 0.05))
```

You can't really trust these results. Haley-Knott regression performs poorly in the case of selective genotyping (as with the `hyper` data). Standard interval mapping or imputation would be better, but Haley-Knott regression has the advantage of speed, which is the reason we use it here.

22. Finally, we consider the fit of multiple-QTL models. Currently, only the use of multiple imputation has been implemented. We first create a QTL object using the function `makeqtl`, with five QTL at specified, fixed positions.

```
chr <- c(1, 1, 4, 6, 15)
pos <- c(50, 76, 30, 70, 20)
qtl <- makeqtl(hyper, chr, pos)
```

Finally, we use the function `fitqtl` to fit a model with five QTL, and allowing the QTL on chromosomes 6 and 15 to interact.

```
my.formula <- y ~ Q1 + Q2 + Q3 + Q4 + Q5 + Q4:Q5
out.fitqtl <- fitqtl(hyper$pheno[,1], qtl, formula=my.formula)
summary(out.fitqtl)
```

23. You may wish to clean up your workspace before we move on to the next example.

```
ls()
rm(list=ls())
```

Example 2: Genetic mapping

R/qtl includes some utilities for estimating genetics maps and checking marker orders. In this example, we describe the use of these utilities.

1. Get access to some sample data. This is simulated data with some errors in marker order.

   ```
   data(badorder)
   summary(badorder)
   plot(badorder)
   ```

2. Estimate recombination fractions between all pairs of markers, and plot them.

   ```
   badorder <- est.rf(badorder)
   plot.rf(badorder)
   ```

   It appears that markers on chromosomes 2 and 3 have been switched.

   Also note that, if we look more closely at the recombination fractions for chromosome 1, there seem to be some errors in marker order.

   ```
   plot.rf(badorder, chr=1)
   ```

3. Re-estimate the genetic map.

   ```
   newmap <- est.map(badorder, verbose=TRUE)
   plot.map(badorder, newmap)
   ```

   This really shows the problems on chromosomes 2 and 3.

4. Fix the problems on chromosomes 2 and 3. First, we look more closely at the recombination fractions for these chromosoems

   ```
   plot.rf(badorder, chr=2:3)
   ```

   We need to move the sixth marker on chromosome 2 to chromosome 3, and the fifth marker on chromosome 3 to chromosome 2. We need to figure out which markers these are.

   ```
   pull.map(badorder, chr=2)
   pull.map(badorder, chr=3)
   ```

   Now we can use the function `movemarker` to move the markers. It seems like they should be exactly switched.

   ```
   badorder <- movemarker(badorder, "D2M937", 3, 48)
   badorder <- movemarker(badorder, "D3M160", 2, 28.8)
   ```

   Now look at the recombination fractions again.

   ```
   plot.rf(badorder, chr=2:3)
   ```

5. We can check the marker order on chromosome 1. The function `ripple` will consider all permutations of a sliding window of adjacent markers. A quick-and-dirty approach is to count the number of obligate crossovers for each possible order, to find the order with the minimum number of crossovers. A more refined, but also more computationally intensive, approach is to re-estimate the genetic map for each order, calculating LOD scores ($\log_{10}$ likelihood ratios) relative to the initial order. (This may be done with allowance for the presence of genotyping errors.) The default approach is the quick-and-dirty method.

   The following checks the marker order on chromosome 1, permuting groups of six contiguous markers.

   ```
   rip1 <- ripple(badorder, chr=1, window=6)
   summary(rip1)
   ```

   In the summary output, markers 9–11 clearly need to be flipped. There also seems to be a problem with the order of markers 4–6.

6. The following performs the likelihood analysis, permuting groups of three adjacent markers, assuming a genotyping error rate of 1%. It's considerably slower, but more trustworthy.

   ```
   rip2 <- ripple(badorder, chr=1, window=3, err=0.01, method="likelihood")
   summary(rip2)
   ```

   Note that positive LOD scores indicate that the alternate order has a higher likelihood than the original.

7. We can switch the order of markers 9–11 with the function `switch.order` (which works only for a single chromosome) and then re-assess the order. Note that the second row of `rip1` corresponds to the improved order.

```
badorder.rev <- switch.order(badorder, 1, rip1[2,])
rip1r <- ripple(badorder.rev, chr=1, window=6)
summary(rip1r)
```

It looks like the marker pairs (5,6) and (1,2) should each be inverted. We use `switch.order` again, and then check marker order using the likelihood method.

```
badorder.rev <- switch.order(badorder.rev, 1, rip1r[2,])
rip2r <- ripple(badorder.rev, chr=1, window=3, err=0.01)
summary(rip2r)
```

It's probably best to start out using the quick-and-dirty method, with a large window size, to find the marker order with the minimum number of obligate crossovers, and then refine that order using the slower, but more trustworthy, likelihood method.

8. We can look again at the recombination fractions for this chromosome.

```
badorder.rev <- est.rf(badorder.rev)
plot.rf(badorder.rev, 1)
```

Example 3: Listeria susceptibility

In order to demonstrate further uses of the function `scanone`, we consider some data on susceptibility to *Listeria monocytogenes* in mice (Boyartchuk et al., Nature Genetics 27:259-260, 2001). These data were kindly provided by Victor Boyartchuk and Bill Dietrich.

1. Get access to the data and view some summaries.

```
data(listeria)
summary(listeria)
plot(listeria)
plot.missing(listeria)
```

Note that in the missing data plot, gray pixels are partially missing genotypes (e.g., a genotype may be known to be either AA or AB, but not which).

The phenotype here is the survival time of a mouse (in hours) following infection with *Listeria monocytogenes*. Individuals with a survival time of 264 hours are those that recovered from the infection.

2. We'll use the log survival time, rather than survival time, so we first need to create a new phenotype, which will end up as the third phenotype (after `sex`).

```
listeria$pheno$logSurv <- log(listeria$pheno[,1])
plot(listeria)
```

3. Estimate pairwise recombination fractions.

```
listeria <- est.rf(listeria)
plot.rf(listeria)
plot.rf(listeria, chr=c(5,13))
```

4. Re-estimate the genetic map.

```
newmap <- est.map(listeria, error.prob=0.01)
plot.map(listeria, newmap)
listeria <- replace.map(listeria, newmap)
```

5. Investigate genotyping errors; nothing gets flagged with a cutoff of 4, but one genotype is indicated with error LOD $\sim$3.6.

```
listeria <- calc.errorlod(listeria, error.prob=0.01)
top.errorlod(listeria)
top.errorlod(listeria, cutoff=3.5)
plot.geno(listeria, chr=13, ind=61:70, cutoff=3.5)
```

Note that in the plot given by `plot.geno`, for an intercross, white = AA, gray = AB, black = BB, green = AA or AB, and orange = AB or BB.

6. Now on to the QTL mapping. Recall that the phenotype distribution shows a clear departure from the standard assumptions for interval mapping; 30% of the mice survived longer than 264 hours, and were considered recovered from the infection.

   One approach for these data is to use the two-part model considered by Boyartchuk et al. (2001). In this model, a mouse with genotype g has probability p_g of surviving the infection. If it does die, its log survival time is assumed to be distributed normal(μ_g,σ^2). Analysis proceeds by maximum likelihood via an EM algorithm. Three LOD scores are calculated. LOD(p, μ) is for the test of the null hypothesis $p_g \equiv p$ and $\mu_g \equiv \mu$. LOD(p) is for the test of the hypothesis $p_g \equiv p$ but the μ are allowed to vary. LOD(μ) is for the test of the hypothesis $\mu_g \equiv \mu$ but the p are allowed to vary.

   The function `scanone` will fit the above model when the argument `model="2part"`. One must also specify the argument `upper`, which indicates whether the spike in the phenotype is the maximum phenotype (as it is with this phenotype; take `upper=TRUE`) or the minimum phenotype (take `upper=FALSE`). For this model, only the EM algorithm has been implemented so far.

   ```
   listeria <- calc.genoprob(listeria, step=2)
   out.2p <- scanone(listeria, pheno.col=3, model="2part", upper=TRUE)
   ```

   Note that, because this model has three extra parameters, the appropriate LOD threshold is higher—around 4.5 rather than 3.5. The three different LOD curves are in columns 3–5 of the output. We can use the `lodcolumn` argument to `plot.scanone` to plot these other LOD scores.

   ```
   summary(out.2p)
   summary(out.2p, threshold=4.5)
   ```

   Alternatively, we may use `format="allpeaks"`, in which case it displays the maximum LOD score or each column, with the position at which each was maximized. You may provide either one threshold, which would be applied to all LOD score columns, or a separate threshold for each column.

   ```
   summary(out.2p, format="allpeaks", threshold=3)
   summary(out.2p, format="allpeaks", threshold=c(4.5,3,3))
   ```

7. By default, `plot.scanone` will plot the first LOD score column. Alternatively, we may indicate another column to plot with the `lodcolumn` argument. Or we can plot up to three LOD scores at once by giving a vector.

   ```
   plot(out.2p)
   plot(out.2p, lodcolumn=2)
   plot(out.2p, lodcolumn=1:3, chr=c(1,5,13,15))
   ```

   Note that the locus on chromosome 1 shows effect mostly on the mean time-to-death, conditional on death; the locus on chromosome 5 shows effect mostly on the probability of survival; and the loci on chromosomes 13 and 15 shows some effect on each.

8. Permutation tests may be performed as before. The output will have three columns, corresponding to the three LOD scores.

   ```
   operm.2p <- scanone(listeria, model="2part", pheno.col=3,
                       upper=TRUE, n.perm=25)
   summary(operm.2p, alpha=0.05)
   ```

   We may again use the permutation results in `summary.scanone` to have thresholds calculated automatically and to obtain genome-scan-adjusted p-values, but of course we would want to have performed more than 25 permutations.

   ```
   summary(out.2p, format="allpeaks", perms=operm.2p,
           alpha=0.05, pvalues=TRUE)
   ```

9. Alternatively, one may perform separate analyses of the log survival time, conditional on death, and the binary phenotype survival/death. First we set up these phenotypes.

   ```
   y <- listeria$pheno$logSurv
   my <- max(y, na.rm=TRUE)
   z <- as.numeric(y==my)
   y[y==my] <- NA
   listeria$pheno$logSurv2 <- y
   listeria$pheno$binary <- z
   plot(listeria)
   ```

We use standard interval mapping for the log survival time conditional on death; the results are slightly different from LOD(μ).

```
out.mu <- scanone(listeria, pheno.col=4)
plot(out.mu, out.2p, lodcolumn=c(1,3), chr=c(1,5,13,15))
```

We can use `scanone` with `model="binary"` to analyze the binary phenotype. Again, the results are only slight different from LOD(p).

```
out.p <- scanone(listeria, pheno.col=5, model="binary")
plot(out.p, out.2p, lodcolumn=c(1,2), chr=c(1,5,13,15))
```

10. A further approach is to use a non-parametric form of interval mapping. R/qtl uses an extension of the Kruskal-Wallis test statistic. Use `scanone` with `model="np"`. In this case, the argument `method` is ignored; the analysis method is much like Haley-Knott regression. If the argument `ties.random=TRUE`, tied phenotypes are ranked at random. If `ties.random=FALSE`, tied phenotypes are given the average rank and a correction is applied to the LOD score.

    ```
    out.np1 <- scanone(listeria, model="np", ties.random=TRUE)
    out.np2 <- scanone(listeria, model="np", ties.random=FALSE)

    plot(out.np1, out.np2)
    plot(out.2p, out.np1, out.np2, chr=c(1,5,13,15))
    ```

    Note that the significance threshold for the non-parametric genome scan will be quite a bit smaller than that for the two-part model. The two approaches for dealing with ties give basically the same results. Randomizing ties for the non-parametric approach can give quite variable results in the case of a great number of ties, and so we would recommend the use of `ties.random=FALSE` in this case.

Example 4: Covariates in QTL mapping

As a further example, we illustrate the use of covariates in QTL mapping. We consider some simulated backcross data.

1. Get access to the data.

    ```
    data(fake.bc)
    summary(fake.bc)
    plot(fake.bc)
    ```

2. Perform genome scans for the two phenotypes without covariates.

    ```
    fake.bc <- calc.genoprob(fake.bc, step=2.5)
    out.nocovar <- scanone(fake.bc, pheno.col=1:2)
    ```

3. Perform genome scans with sex as an additive covariate. Note that the covariates must be numeric. Factors may have to be converted.

    ```
    sex <- fake.bc$pheno$sex
    out.acovar <- scanone(fake.bc, pheno.col=1:2, addcovar=sex)
    ```

    Here, the average phenotype is allowed to be different in the two sexes, but the effect of the putative QTL is assumed to be the same in the two sexes.

4. Note that the use of sex as an additive covariate resulted in an increase in the LOD scores for phenotype 1, but resulted in a decreased LOD score at the chr 5 locus for phenotype 2.

    ```
    summary(out.nocovar, threshold=3, format="allpeaks")
    summary(out.acovar, threshold=3, format="allpeaks")

    plot(out.nocovar, out.acovar, chr=c(2, 5))
    plot(out.nocovar, out.acovar, chr=c(2, 5), lodcolumn=2)
    ```

5. Let us now perform genome scans with sex as an interactive covariate, so that the QTL is allowed to be different in the two sexes.

    ```
    out.icovar <- scanone(fake.bc, pheno.col=1:2, addcovar=sex, intcovar=sex)
    ```

6. The LOD score in the output is for the comparison of the full model with terms for sex, QTL and QTL×sex interaction to the reduced model with just the sex term. Thus, the degrees of freedom associated with the LOD score is 2 rather than 1, and so larger LOD scores will generally be obtained.

```
summary(out.icovar, threshold=3, format="allpeaks")
```

```
plot(out.acovar, out.icovar, chr=c(2,5), col=c("blue", "red"))
plot(out.acovar, out.icovar, chr=c(2,5), lodcolumn=2,
     col=c("blue", "red"))
```

7. The difference between the LOD score with sex as an interactive covariate and the LOD score with sex as an additive covariate concerns the test of the QTL×sex interaction: does the QTL have the same effect in both sexes? The differences, and a plot of the differences, may be obtained as follows.

```
out.sexint <- out.icovar - out.acovar
plot(out.sexint, lodcolumn=1:2, chr=c(2,5), col=c("green", "purple"))
```

   The green and purple curves are for the first and second phenotypes, respectively.

8. To test for the QTL×sex interaction, we may perform a permutation test. This is not perfect, as the permutation test eliminates the effect of the QTL, and so we must assume that the distribution of the LOD score for the QTL×sex interaction is the same in the presence of a QTL as under the global null hypothesis of no QTL effect.

   The permutation test requires some care. We must perform separate permutations with sex as an additive covariate and with sex as an interactive covariate, but we must ensure, by setting the "seed" for the random number generator, that they use matched permutations of the data.

   For the sake of speed, we will use Haley-Knott regression, even though the results above were obtained by standard interval mapping. Also, we will perform just 100 permutations, though 1000 would be preferred.

```
seed <- ceiling(runif(1, 0, 10^8))
set.seed(seed)
operm.acovar <- scanone(fake.bc, pheno.col=1:2, addcovar=sex,
                        method="hk", n.perm=100)
set.seed(seed)
operm.icovar <- scanone(fake.bc, pheno.col=1:2, addcovar=sex,
                        intcovar=sex, method="hk", n.perm=100)
```

   Again, the differences concern the QTL×sex interaction.

```
operm.sexint <- operm.icovar - operm.acovar
```

   We can use `summary` to get the genome-wide LOD thresholds.

```
summary(operm.sexint, alpha=c(0.05, 0.20))
```

   We can also use these results to look at evidence for QTL×sex interaction in our initial scans.

```
summary(out.sexint, perms=operm.sexint, alpha=0.1,
        format="allpeaks", pvalues=TRUE)
```

Internal data structure

Finally, let us briefly describe the rather complicated data structure that R/qtl uses for QTL mapping experiments. This will be rather dull, and will require a good deal of familiarity with the R (or S) language. The choice of data structure 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 become extremely complicated.

1. Get access to some sample data.

```
data(fake.bc)
```

2. 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`.

```
class(fake.bc)
```

3. Every `cross` object has two components, one containing the genotype data and genetic maps and the other containing the phenotype data.

```
names(fake.bc)
```

4. The phenotype data is simply a matrix (more strictly a data.frame) with rows corresponding to individuals and columns corresponding to phenotypes.

```
fake.bc$pheno
```

5. 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 either `"A"` or `"X"`, according to whether it is an autosome or the X chromosome.

```
names(fake.bc$geno)
sapply(fake.bc$geno, class)
```

6. Each component of `geno` contains two components, `data` (containing the marker genotype data) and `map` (containing the positions of the markers, in cM).

```
names(fake.bc$geno[[3]])
fake.bc$geno[[3]]$data[1:5,]
fake.bc$geno[[3]]$map
```

That's it for the raw data.

7. When one runs `calc.genoprob`, `sim.geno`, `argmax.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(fake.bc$geno[[3]])
fake.bc <- calc.genoprob(fake.bc, step=10, err=0.01)
names(fake.bc$geno[[3]])
fake.bc <- sim.geno(fake.bc, step=10, n.draws=8, err=0.01)
names(fake.bc$geno[[3]])
fake.bc <- argmax.geno(fake.bc, step=10, err=0.01)
names(fake.bc$geno[[3]])
fake.bc <- calc.errorlod(fake.bc, err=0.01)
names(fake.bc$geno[[3]])
```

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

```
names(fake.bc)
fake.bc <- est.rf(fake.bc)
names(fake.bc)
```