A brief tour of R/qtl

Karl W Broman

Department of Biostatistics, Johns Hopkins University
http://www.biostat.jhsph.edu/~kbroman/qtl

13 June 2003

Overview of R/qtl

R/qtl is an extensible, interactive environment for mapping quantitative trait loci (QTLs) 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). The fit of higher-order QTL models, with sophisticated techniques for model comparison and model search, will be incorporated soon.

R/qtl is distributed as source code for Unix or compiled code for Windows or MacOS. 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 'GNU S'—A language and environment for statistical computing and graphics. R is similar to the award-winning S system, which was developed at Bell Laboratories by John Chambers et al. It provides a wide variety of statistical and graphical techniques (linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, ...).
>
> R is designed as a true computer language with control-flow constructions for iteration and alternation, and it allows users to add additional functionality by defining new functions. For computationally intensive tasks, C, C++ and Fortran code can be linked and called at run time.

R is freely available for Windows, Unix and MacOS, 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. For example, see WN Venables, BD Ripley (2002) *Modern Applied Statistics with S*, 4th edition. Springer.

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 modelled 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.info	Plot the proportion of missing genotype data
	summary.cross	Print summary of QTL experiment
	nchr, nind, nmar, nphe, totmar	
Data manipulation	c.cross	Combine crosses
	clean	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 multiple 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
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
	max.scanone	Maximum LOD score in one-dimensional genome scan
	max.scantwo	Maximum LOD score in two-dimensional genome scan
	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 only for 1 marker
Genetic mapping	est.map	Estimate genetic maps
	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
	plot.errorlod	Plot grid of error LOD values
	plot.geno	Plot observed genotypes, flagging likely errors
	top.errorlod	List genotypes with highest error LOD values
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
	scan.qtl	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 or Splus, 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 websites cited above. We assume here that the user is running a version of the Windows operating system.

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 click (on the menu bar) Help → R language (html). 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. Typing the following will make the above commands open the html version of help.

   ```
   options(htmlhelp=TRUE)
   ```

   You can create a file `"c:\.Rprofile"` containing any R code to be executed whenever R is started. The commands `library(qtl)` and `options(htmlhelp=TRUE)` are good candidates for placement in such a file.

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)
   hist(hyper$pheno[,1], breaks=30)
   ```

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,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,6)
   plot.missing(hyper,6)
   ```

7. For some functions that require a good deal of computation time, trace information is printed so that the user may be made aware of the progress of the calculations. In R for Windows, one needs to turn off "buffered output" in order to view this trace information. Either type the shortcut key Ctrl+W or click (on the menu bar) Misc→ Buffered output.

8. 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, trace=TRUE)
   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)
```

9. 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.

   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, we must assume no crossover interference. The function `calc.genoprob` performs the first of these; the values are used in the calculation of the error LOD scores.

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

   We may now make various plots and receive other summary information regarding which genotypes are likely in error.

   ```
   plot.errorlod(hyper)
   top.errorlod(hyper)
   plot.errorlod(hyper, chr=c(4,11,16))
   ```

10. 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=71:90, min.sep=4)
    ```

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

11. 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`.) The function makes use of the results from `calc.genoprob`, and so we first calculate the genotype probabilities at 2 cM steps.

    ```
    hyper <- calc.genoprob(hyper, step=2)
    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")
    ```

12. We now, finally, get to QTL mapping. The QTL genotype probabilities must first be calculated, using `calc.genoprob`, but we did that above. We use the function `scanone` to perform a single-QTL genome scan with a normal model. We may use maximum likelihood via the EM algorithm 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), but the method is somewhat time and memory consuming, and so we will not explore the method here.

13. 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, 3)
    summary(out.hk, 3)
    ```

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

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

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

16. The function `scanone` may also be used to perform a permutation test to get a genome-wide LOD significance threshold. This can take a long time, so we'll just do a few permutations.

```
operm.hk <- scanone(hyper, method="hk", n.perm=25)
quantile(operm.hk, 0.95)
```

17. 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()
```

18. The function `scantwo` performs a two-dimensional genome scan with a two-QTL model. For every pair of positions, it calculates a joint LOD score and a LOD score for a test of epistasis. This can be quite time consuming, and so you may wish to do the calculations on a coarser grid.

```
hyper.coarse <- calc.genoprob(hyper, step=10, error.prob=0.01)
out2.hk <- scantwo(hyper.coarse, method="hk")
out2.em <- scantwo(hyper.coarse, method="em")
```

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

```
summary(out2.hk, c(8,3,3))
summary(out2.hk, c(0,4,Inf))
summary(out2.hk, c(0,Inf,4))
```

The summary function requires LOD thresholds on the joint LOD score, the epistasis LOD score, and on the conditional LOD score for a single QTL. For each pair of chromosomes, the locus pair with the maximum joint LOD is printed if the joint LOD exceeds its threshold and either the epistasis LOD score exceeds its threshold or both loci have conditional LOD scores that exceed their threshold.

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

In the plot of the results of `scantwo`, the epistasis LOD scores appear in the upper left triangle and the joint LOD scores appear in the lower right triangle. 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 position with the maximum joint LOD score, and that with the maximum epistatic LOD score.

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

21. One may also use `scantwo` to perform permutation tests in order to obtain genome-wide LOD significance thresholds. These may take days to complete, so we'll just do three permutations.

```
operm2 <- scantwo(hyper.coarse, method="hk", n.perm=3)
apply(operm2, 2, quantile, 0.95)
```

The output of `scantwo`, when `n.perm` is specified, is a matrix with two columns—the maximum joint and epistasis LOD scores, across the two-dimensional genome scan, for each permutation replicate.

22. 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. Unfortunately, R/qtl doesn't have facilities for easily fixing such problems. It might be best to work directly with the raw data and then import it again.

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, trace=TRUE)
plot.map(badorder, newmap)
```

This really shows the problems on chromosomes 2 and 3.

4. 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.

5. 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.

6. 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.

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

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

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=10, 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)
   ```