R/qtl tutorial

Karl W Broman

Department of Biostatistics and Medical Informatics
University of Wisconsin – Madison

http://www.rqtl.org

30 November 2012

Preliminaries

1. To install R/qtl, type (within R)

   ```
   install.packages("qtl")
   ```

   (This needs to be done just once.)

2. To load the R/qtl package, type

   ```
   library(qtl)
   ```

   This needs to be done every time you start R. (There is a way to have the package loaded automatically every time, but we won't discuss that here.)

3. To view the objects in your workspace:

   ```
   ls()
   ```

4. To get information on a function or data set in R (and in R/qtl) use the function `help()`, or the shortcut, `?`, as follows:

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

5. 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/rqtltour3.R")
   ```

6. I recommend using RStudio instead of the R graphical user interface; it is available for Windows, Mac and Unix at `http://www.rstudio.com/ide`.

Data import

We will consider data from Sugiyama et al., Physiol Genomics 10:5–12, 2002. The data are from an intercross between the BALB/cJ and CBA/CaJ mouse strains. Only male offspring were considered. There are four phenotypes: blood pressure, heart rate, body weight, and heart weight. We will focus on the blood pressure phenotype, will consider just the 163 individuals with genotype data and, for simplicity, will focus on the autosomes. The data are contained in the comma-delimited file `http://www.rqtl.org/sug.csv`.

7. Load the data into R/qtl as follows.

   ```
   sug <- read.cross("csv", "http://www.rqtl.org", "sug.csv",
                     genotypes=c("CC", "CB", "BB"), alleles=c("C", "B"))
   ```

The function `read.cross()` is for importing data into R/qtl. `"sug.csv"` is the name of the file, which we import directly from the R/qtl website. (If the file is stored on your computer, you can replace the website with the directory, using forward slashes (`/`) in place of any backslashes (`\`).) `genotypes` indicates the codes used for the genotypes; `alleles` indicates single-character codes to be used in plots and such.

`read.cross()` loads the data from the file and formats it into a special `"cross"` object, which is then assigned to `sug` via the assignment operator (`<-`).

Diagnostics

Generally, at this point, one would spend considerable time studying the genotype and phenotype data, looking for potential errors. In many cases, about half of the analysis time is devoted to such diagnostics. We'll postpone this until a bit later.

Summaries

The data object `sug` is complex; it contains the genotype data, phenotype data and genetic map. R has a certain amount of "object oriented" facilities, so that calls to functions like `summary()` and `plot()` are interpreted appropriately for the object considered.
The object `sug` has "class" `"cross"`, and so calls to `summary()` and `plot()` are sent to the functions `summary.cross()` and `plot.cross()`.

8. Get a quick summary of the data. (This also performs a variety of checks of the integrity of the data.)

   ```
   summary(sug)
   ```

   We see that this is an intercross with 163 individuals. There are 6 phenotypes, and genotype data at 93 markers across the 19 autosomes. The genotype data is quite complete.

9. There are a number of simple functions for pulling out pieces of summary information. Hopefully these are self-explanatory.

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

10. Get a summary plot of the data.

    ```
    plot(sug)
    ```

    The plot in the upper-left shows the pattern of missing genotype data, with black pixels corresponding to missing genotypes. The next plot shows the genetic map of the typed markers. The following plots are histograms or bar plots for the six phenotypes. The last two "phenotypes" are sex (with 1 corresponding to males) and mouse ID.

11. Individual parts of the above plot may be obtained as follows.

    ```
    plotMissing(sug)
    plotMap(sug)
    plotPheno(sug, pheno.col=1)
    plotPheno(sug, pheno.col=2)
    plotPheno(sug, pheno.col=3)
    plotPheno(sug, pheno.col=4)
    plotPheno(sug, pheno.col=5)
    plotPheno(sug, pheno.col=6)
    ```

12. You can also refer to phenotypes by name, as follows.

    ```
    plotPheno(sug, pheno.col="bp")
    plotPheno(sug, pheno.col="bw")
    ```

Single-QTL analysis

Let's now proceed to QTL mapping via a single-QTL model.

13. We first calculate the QTL genotype probabilities, given the observed marker data, via the function `calc.genoprob()`. This is done at the markers and at a grid along the chromosomes. The argument `step` is the density of the grid (in cM), and defines the density of later QTL analyses.

    ```
    sug <- calc.genoprob(sug, step=1)
    ```

    The output of `calc.genoprob()` is the same cross object as input, with additional information (the QTL genotype probabilities) inserted. We assign this back to the original object (writing over the previous data), though it could have also been assigned to a new object.

14. To perform a single-QTL genome scan, we use the function `scanone()`. By default, it performs standard interval mapping (that is, maximum likelihood via the EM algorithm). Also, by default, it considers the first phenotype in the input cross object (in this case, blood pressure).

    ```
    out.em <- scanone(sug)
    ```

15. The output has "class" `"scanone"`. The `summary()` function is passed to the function `summary.scanone()`, and gives the maximum LOD score on each chromosome.

    ```
    summary(out.em)
    ```

16. Alternatively, we can give a threshold, e.g., to only see those chromosomes with LOD > 3.

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

17. We can plot the results as follows.

    ```
    plot(out.em)
    ```

18. We can do the genome scan via Haley-Knott regression by calling `scanone()` with the argument `method="hk"`.

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

19. We may plot the two sets of LOD curves together in a single call to `plot()`.

    ```
    plot(out.em, out.hk, col=c("blue", "red"))
    ```

20. Alternatively, we could do the following:

    ```
    plot(out.em, col="blue")
    plot(out.hk, col="red", add=TRUE)
    ```

21. We can plot the LOD curves for just chromosomes 7 and 15 as follows.

    ```
    plot(out.em, out.hk, col=c("blue", "red"), chr=c(7,15))
    ```

22. In the alternate approach to this, we need to indicate chromosomes in both lines:

    ```
    plot(out.em, col="blue", chr=c(7,15))
    plot(out.hk, col="red", chr=c(7,15), add=TRUE)
    ```

23. It is perhaps more informative to plot the differences:

    ```
    plot(out.hk - out.em, ylim=c(-0.3, 0.3), ylab="LOD(HK)-LOD(EM)")
    ```

Permutation tests

To perform a permutation test, to get a genome-wide significance threshold or genome-scan-adjusted p-values, we use the function `scanone()` just as before, but with an additional argument, `n.perm`, indicating the number of permutation replicates. It is quickest to use Haley-Knott regression.

24. In case the time to perform the permutation test is too long, you can skip it (here) and load the results (that I calculated previously) for this plus other time-consuming stuff we'll see shortly as follows.

    ```
    load(url("http://www.rqtl.org/various.RData"))
    ```

25. The code to do the actual permutation test is the following:

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

26. A histogram of the results (the 1000 genome-wide maximum LOD scores) is obtained as follows:

    ```
    plot(operm)
    ```

27. Significance thresholds may be obtained via the `summary()` function:

    ```
    summary(operm)
    summary(operm, alpha=c(0.05, 0.2))
    ```

28. Most importantly, the permutation results may be used along with the `scanone()` results to have significance thresholds and p-values calculated automatically:

    ```
    summary(out.hk, perms=operm, alpha=0.2, pvalues=TRUE)
    ```

Interval estimates of QTL location

For the blood pressure phenotype, we've seen good evidence for QTL on chromosomes 7 and 15. Interval estimates of the location of QTL are commonly obtained via 1.5-LOD support intervals, which may be calculated via the function `lodint()`. Alternatively, an approximate Bayes credible interval may be obtained with `bayesint()`.

29. To obtain the 1.5-LOD support interval and 95% Bayes interval for the QTL on chromosome 7, type:

    ```
    lodint(out.hk, chr=7)
    bayesint(out.hk, chr=7)
    ```

    The first and last rows define the ends of the intervals; the middle row is the estimated QTL location.

30. It is sometimes useful to identify the closest flanking markers; use `expandtomarkers=TRUE`:

    ```
    lodint(out.hk, chr=7, expandtomarkers=TRUE)
    bayesint(out.hk, chr=7, expandtomarkers=TRUE)
    ```

31. We can calculate the 2-LOD support interval and the 99% Bayes interval as follows.

    ```
    lodint(out.hk, chr=7, drop=2)
    bayesint(out.hk, chr=7, prob=0.99)
    ```

32. The intervals for the chr 15 locus may be calculated as follows.

    ```
    lodint(out.hk, chr=15)
    bayesint(out.hk, chr=15)
    ```

QTL effects

We may obtain plots indicating the estimated effects of the QTL via `plotPXG()`, which creates a dot plot, or `effectplot()`, which plots the average phenotype for each genotype group.

33. For `plotPXG()`, we must first identify the marker closest to the QTL peak. Use `find.marker()`.

    ```
    max(out.hk)
    mar <- find.marker(sug, chr=7, pos=47.7)
    plotPXG(sug, marker=mar)
    ```

    Note that red dots correspond to inferred genotypes (based on a single imputation).

34. The function `effectplot()` uses multiple imputation to account for missing genotypes. First call `sim.geno()`, which generates the imputated genotypes. The argument `n.draws` indicates the number of imputations.

    ```
    sug <- sim.geno(sug, n.draws=64, step=1)
    effectplot(sug, mname1=mar)
    ```

35. We may use `effectplot()` at a position on the "grid" between markers, using `"7@47.7"` to indicate the position at 47.7 cM on chr 7.

    ```
    effectplot(sug, mname1="7@47.7")
    ```

36. Similar plots may be obtained for the locus on chr 15.

    ```
    max(out.hk, chr=15)
    mar2 <- find.marker(sug, chr=15, pos=12)
    plotPXG(sug, marker=mar2)
    effectplot(sug, mname1="15@12")
    ```

37. We may plot the joint effects of the two loci via `plotPXG()` as follows. It is often useful to look at it in both ways.

    ```
    plotPXG(sug, marker=c(mar, mar2))
    plotPXG(sug, marker=c(mar2, mar))
    ```

38. The function `effectplot()` gives more readable figures in this case.

    ```
    effectplot(sug, mname1="7@47.7", mname2="15@12")
    effectplot(sug, mname2="7@47.7", mname1="15@12")
    ```

    The two loci do not appear to interact.

Other phenotypes

By default in `scanone()`, we consider the first phenotype in the input cross object. Other phenotypes, include the parallel consideration of multiple phenotypes, can be considered via the argument `pheno.col`.

39. To analyze the second phenotype, refer to it by its numeric index, as follows.

    ```
    out.hr <- scanone(sug, pheno.col=2, method="hk")
    ```

40. Alternatively, refer to a phenotype by its name:

    ```
    out.bw <- scanone(sug, pheno.col="bw", method="hk")
    ```

41. You can also give a numeric vector of phenotype values. This is useful for considering a transformed version of a phenotype, such as log body weight.

    ```
    out.logbw <- scanone(sug, pheno.col=log(sug$pheno$bw), method="hk")
    ```

42. Use of a vector of phenotype indices results in an object with multiple LOD score columns, one for each phenotype.

    ```
    out.all <- scanone(sug, pheno.col=1:4, method="hk")
    ```

43. For this final case, it's important to note that the `summary()` function, by default, focuses solely on the first LOD score column.

    ```
    summary(out.all, threshold=3)
    ```

    Here, it looks at the first LOD score column and picks off the peaks that are above 3, and then gives the LOD scores at that location for the other three columns.

    To do the same thing but focusing on another column, use the argument `lodcolumn`.

    ```
    summary(out.all, threshold=3, lodcolumn=4)
    ```

44. Alternatively, use `format="allpeaks"`, to get the maximum LOD score for each column, with a chromosome being shown if at least one of the LOD score column exceeds the threshold.

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

45. A third version of the output is obtained with `format="allpheno"`, which gives one row per LOD peak and gives the LOD scores for all columns at each peak.

    ```
    summary(out.all, threshold=3, format="allpheno")
    ```

46. There are two other formats that might be preferred: `format="tabByCol"` and `format="tabByChr"`. These give tables with one significant LOD peak per phenotype, organized either by phenotype (with `"tabByCol"`) or by chromosome (with `"tabByChr"`). The tables include 1.5-LOD support intervals, and so one may wish to use `"tabByCol"` even if there is only one LOD score column.

    ```
    summary(out.all, threshold=3, format="tabByCol")
    summary(out.all, threshold=3, format="tabByChr")
    ```

Data diagnostics

To illustrate the process of data cleaning, we will consider a simulated data set, an intercross with 250 individuals. The data are contained in the comma-delimited file http://www.rqtl.org/bad.csv.

47. Load the data into R/qtl as follows.

    ```
    bad <- read.cross("csv", "http://www.rqtl.org", "bad.csv")
    ```

48. Look at a quick summary.

    ```
    summary(bad)
    ```

49. We first consider the three phenotypes. We start with histograms. The function `par(mfrow=c(3,1))` is used to make a figure with three panels.

    ```
    par(mfrow=c(3,1))
    for(i in 1:3)
      plotPheno(bad, pheno.col=i)
    ```

    Note that the third phenotype has some clear outliers. Often such problems arise from data entry errors.

50. It is often useful to plot the phenotypes as a function of the time of measurement, or by individual index. This can reveal batch effects. (Ideally, if phenotyping were performed in batches, such batches would have been recorded, and we would evaluate them directly.)

    ```
    par(mfrow=c(3,1))
    for(i in 1:3)
      plot(bad$pheno[,i], ylab=names(bad$pheno)[i])
    ```

    The third phenotype again shows a problem: individuals 150–200 or so are shifted downward relative to others.

51. Finally, scatterplots of the phenotypes against one another may reveal problems. The R function `pairs()` provides a matrix of all pairs of scatterplots.

    ```
    par(mfrow=c(1,1))
    pairs(bad$pheno)
    ```

    Here we see a problem with the first and second phenotypes: a pair of outlying points. These may be recording errors.

52. Turning to the genotype data, we first look for markers and individuals with excessive missing data. We would generally omit such markers and individuals, at least initially, as they often indicate problems.

    ```
    par(mfrow=c(2,1))
    plot(ntyped(bad, "ind"), main="No. genotypes, by ind'l",
                ylab="No. genotypes")
    plot(ntyped(bad, "mar"), main="No. genotypes, by marker",
                ylab="No. genotypes")
    ```

    There are two individuals and three markers with a great deal of missing data. To omit them, we do the following:

    ```
    bad <- subset(bad, ind=(ntyped(bad, "ind") > 100))
    nt.mar <- ntyped(bad, "mar")
    bad <- drop.markers(bad, names(nt.mar)[nt.mar < 180])
    ```

53. We next look for pairs of individuals with similar genotypes, which may arise from problems in the handling of DNA samples (duplicate samples). We use the function `comparegeno()`, which calculates, for each pair of individuals, the proportion of matching genotypes.

    ```
    cg <- comparegeno(bad)
    lowertri <- cg[lower.tri(cg)]
    par(mfrow=c(1,1))
    hist(lowertri, breaks=seq(0, 1, by=0.01))
    rug(lowertri)
    ```

    The function `lower.tri()` is used to pull out the lower triangle from the matrix of results, giving one value per pair of individuals. The function `rug()` plots tick marks at the data points along the bottom of the histogram.

    Note that, to identify such sample duplicates, one needs a relatively large genome and many typed markers. We want to see a separation between the distribution for the "good" pairs and the duplicates.

To look at the proportion of matching genotypes for the top ten pairs, do the following.

```
sort(lowertri, decreasing=TRUE)[1:10]
```

54. There are a couple of pairs of individuals with extremely similar genotypes. Without further information, we should omit all four of these individuals, as we don't know which phenotypes attach to the genotypes.

```
wh <- which(!is.na(cg) & cg > 0.9, arr.ind=TRUE)
bad <- subset(bad, ind = -wh[,2])
```

55. We next look for possible segregation distortion: Are the marker genotypes in the proportions 1:2:1? We use the function `geno.table()` to assess 1:2:1 segregation via a χ^2 test, and pull out the markers with significant discrepancy, after a Bonferroni adjustment for the number of tests.

```
gt <- geno.table(bad)
gt[gt$P.value < 0.05/nrow(gt),]
```

There are discrepancies for several markers on chromosome 3 plus one each on chromosomes 6, 7, and 12.

56. We can also look at plots of these results, using the argument `scanone.output=TRUE`. The function `abline()` is used to add horizontal lines; `legend()` adds a legend.

```
gt2 <- geno.table(bad, scanone.output=TRUE)
par(mfrow=c(2,1))
plot(gt2)
plot(gt2, lod=3:5, ylab="Genotype frequency")
abline(h=c(0.25, 0.5), lty=2, col="green")
legend("bottomleft", colnames(gt2)[5:7], lwd=2,
       col=c("black","blue","red"))
```

57. The pattern on chromosome 3 looks like real segregation distortion, and so this can be ignored.

```
geno.table(bad, chr=3)
```

58. The other three markers look like genotyping problems.

```
geno.table(bad, chr=6)
```

59. Let us delete these three markers, as follows:

```
gt3 <- geno.table(bad, chr=-3)
badmar <- rownames(gt3)[gt3$P.value < 0.05/nrow(gt3)]
bad <- drop.markers(bad, badmar)
```

60. Next, we investigate potential problems in marker order and chromosome assignment. First, we plot the estimated recombination fractions between all marker pairs. The function `est.map()` estimates the recombination fractions as well as LOD scores for a test of rf = 1/2. These are stored within the cross object, and then `plotRF()` plots them.

```
bad <- est.rf(bad)
par(mfrow=c(1,1))
plotRF(bad)
```

The upper-left triangle contains the estimated recombination fractions; the lower-right triangle contains the LOD scores. Red indicates linkage (small recombination fraction and large LOD score); blue indicates no linkage (recombination fractions near 1/2 and small LOD scores).

61. There appears to be a problem with a marker on chromosome 7, which appears to be linked to markers on chromosome 10. We can plot just these two chromosomes as follows:

```
plotRF(bad, chr=c(7,10))
```

It looks like the fourth marker on chromosome 7 belongs on chromosome 10.

62. To identify the marker with the problem, we can use `markernames()`.

```
markernames(bad, chr=7)
markernames(bad, chr=7)[4]
```

63. It is also useful to re-estimate the inter-marker distances, using `est.map()`, and plot the map contained in the cross object next to the re-estimated map.

    ```
    newmap <- est.map(bad)
    par(mfrow=c(1,1))
    plotMap(bad, newmap)
    ```

    The problem on chromosome 7 is striking. We can plot just that chromosome, with marker names, to identify the problem marker.

    ```
    plotMap(bad, newmap, chr=7, show.marker.names=TRUE)
    ```

64. At this point, we could either drop the marker, using `drop.markers()`, or we could try to find where the marker maps on chromosome 10, using `tryallpositions()`. Let us try the latter.

    ```
    out <- tryallpositions(bad, "D7M4", chr=10)
    summary(out)
    ```

65. This gives strong evidence that the marker belongs on chromosome 10, between markers D10M1 and D10M2. We can use `movemarker()` to move the marker to that position.

    ```
    bad <- movemarker(bad, "D7M4", 10, 9.245)
    ```

66. We then re-estimate the map and plot it with that contained in the cross.

    ```
    newmap <- est.map(bad)
    plotMap(bad, newmap)
    ```

67. Now we see a potential problem on chromosome 18. Let us look at the recombination fractions for that chromosome.

    ```
    plotRF(bad, chr=18)
    ```

    It is hard to see what is going on, but it may be that the third marker belongs closer to the distal end of the chromosome.

68. We can use `ripple()` to investigate alternate marker orders for chromosome 18.

    ```
    rip <- ripple(bad, chr=18, window=7)
    summary(rip)
    ```

    This indicates strong evidence for moving the third marker to the interval between the sixth and seventh markers.

69. We can use `compareorder()` to compare the likelihoods of those two orders (the original and that with the third marker moved to between the sixth and seventh).

    ```
    compareorder(bad, chr=18, rip[2,])
    ```

    There is very strong evidence for the change in order.

70. We can also use `tryallpositions()` again.

    ```
    mar <- markernames(bad, chr=18)[3]
    out <- tryallpositions(bad, mar, 18)
    summary(out)
    ```

71. Let us move the marker to the estimated position and look at the re-estimated map again.

    ```
    bad <- movemarker(bad, mar, 18, 53.7)
    newmap <- est.map(bad)
    plotMap(bad, newmap)
    ```

    Things are looking fine.

72. Another useful diagnostic is to look at counts of inferred crossovers in each individual. Individuals with unusually large numbers of crossovers are indicated to have genotype problems.

    ```
    xo <- countXO(bad)
    hist(xo, breaks=50)
    rug(xo)
    ```

    The bulk of individuals have 15–40 crossovers, but one individual shows >70 crossovers.

73. Let us omit the individuals with excessive crossovers.

    ```
    bad <- subset(bad, ind=(xo < 60))
    ```

74. Finally, we may investigate possible genotyping errors. We calculate the error LOD scores of Lincoln and Lander (1992). A LOD score is calculated for each individual at each marker; large error LOD scores indicate likely genotyping errors. We use `calc.errorlod()` to calculate the error LOD scores and `top.errorlod()` to view the potential problems.

    ```
    bad <- calc.errorlod(bad, err=0.01)
    top.errorlod(bad)
    ```

    No potential genotyping errors are found.

Two-dimensional, two-QTL scans

Two-dimensional, two-QTL scans offer the opportunity to detect interacting loci or to separate pairs of linked QTL. Analysis is performed with `scantwo()`, which is much like `scanone()`.

75. We will return to the dataset from Sugiyama et al. (2002). If necessary, we re-load the data using `read.cross()`.

    ```
    sug <- read.cross("csv", "http://www.rqtl.org", "sug.csv",
                      genotypes=c("CC", "CB", "BB"), alleles=c("C", "B"))
    ```

    We may also need to re-load the `various.RData` file.

    ```
    load(url("http://www.rqtl.org/various.RData"))
    ```

76. For two-dimensional scans, it's advantageous to run things at a coarser step size, by first re-running `calc.genoprob()`.

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

77. To perform a two-dimensional scan for the blood pressure phenotype, use the following. Note that if you had loaded `"various.RData"` above, you can skip this, as you already have the results.

    ```
    out2 <- scantwo(sug, method="hk")
    ```

78. We may plot the results as follows.

    ```
    plot(out2)
    ```

    The upper-triangle contains interaction LOD scores, comparing the full two-locus model to the additive two-locus model. The lower-triangle contains the "full" LOD scores, comparing the full two-locus model to the null model. Because of the clear evidence for QTL on chromosomes 7 and 15, we see "tails" along those two chromosomes: the two locus model with either chr 7 or chr 15 and anything else is clearly better than the null model.

79. It is best to replace the lower-triangle with the LOD score comparing the full model to the best single-QTL model, using either `lower="cond-int"` or `lower="fv1"` (the two are equivalent).

    ```
    plot(out2, lower="fv1")
    ```

80. We can also look at the LOD scores comparing the additive two-QTL model to the best single-QTL model, using either `upper="cond-add"` or `upper="av1"`.

    ```
    plot(out2, lower="fv1", upper="av1")
    ```

81. To assess significance, we need to do a permutation test. This can be extremely time consuming. The results were already loaded in step 24 on page 3, but here is the code (though here I cite `n.perm=5` rather than `n.perm=1000`, as I'd recommend).

    ```
    operm2 <- scantwo(sug, method="hk", n.perm=5)
    ```

82. With the permutation results in hand, we can get a summary with p-values.

    ```
    summary(out2, perms=operm2, alpha=0.2, pvalues=TRUE)
    ```

    The pair of loci on 7 and 15 are clear. They show no evidence for an interaction. There is some evidence for an additional locus on chr 12, with p=0.17.

Multiple-QTL analyses

After performing the single- and two-QTL genome scans, it's best to bring the identified loci together into a joint model, which we then refine and from which we may explore the possibility of further QTL. In this effort, we work with "QTL objects" created by `makeqtl()`. We fit multiple-QTL models with `fitqtl()`. A number of additional functions will be introduced below.

83. Let's re-run `calc.genoprob()` so that we are working at a step size of 1 cM again.

    ```
    sug <- calc.genoprob(sug, step=1)
    ```

84. First, we create a QTL object containing the loci on chr 7 and 15.

    ```
    qtl <- makeqtl(sug, chr=c(7,15), pos=c(47.7, 12), what="prob")
    ```

    The last argument, `what="prob"`, indicates to pull out the QTL genotype probabilities for use in Haley-Knott regression.

85. We fit the two-locus additive model as follows.

    ```
    out.fq <- fitqtl(sug, qtl=qtl, method="hk")
    summary(out.fq)
    ```

    A key part of the output is the "drop one term at a time" table, which compares the fit of the two-QTL model to the reduced models in which a single QTL is omitted.

86. We may obtain the estimated effects of the QTL via `get.ests=TRUE`. We use `dropone=FALSE` to suppress the drop-one-term analysis.

    ```
    summary(fitqtl(sug, qtl=qtl, method="hk", get.ests=TRUE, dropone=FALSE))
    ```

    Since this is an intercross, we obtain estimates of the additive effect and dominance deviation for each locus.

87. To assess the possibility of an interaction between the two QTL, we may fit the model with the interaction, indicated via a model "formula". The QTL are referred to as `Q1` and `Q2` in the formula, and we may indicate the interaction in a couple of different ways.

    ```
    out.fqi <- fitqtl(sug, qtl=qtl, method="hk", formula=y~Q1*Q2)
    out.fqi <- fitqtl(sug, qtl=qtl, method="hk", formula=y~Q1+Q2+Q1:Q2)
    summary(out.fqi)
    ```

    There is no evidence for an interaction.

88. Another way to assess interactions is with the function `addint()`, which adds one interaction at a time, in the context of a multiple-QTL model. This is most useful when there are more than two QTL being considered.

    ```
    addint(sug, qtl=qtl, method="hk")
    ```

89. The locations of the two QTL are as estimated via the single-QTL scan. We may refine our estimates of QTL location in the context of the multiple-QTL model via `refineqtl()`. This function uses a "greedy" algorithm to iteratively refine the locations of the QTL, one at a time, at each step seeking to improve the overall fit.

    ```
    rqtl <- refineqtl(sug, qtl=qtl, method="hk")
    rqtl
    ```

    The location of each QTL changed slightly, as did the overall LOD score.

90. We can re-run `fitqtl()` to get the revised drop-one-term table.

    ```
    summary(out.fqr <- fitqtl(sug, qtl=rqtl, method="hk"))
    ```

91. The function `plotLodProfile()` plots LOD profiles obtained during the call to `refineqtl()`. These are one-dimensional summaries of the precision of QTL localization, in the context of the multiple-QTL model.

    ```
    plotLodProfile(rqtl)
    ```

    For each position on the curve for the chr 7 QTL, we compare the two-QTL model with the chr 7 locus in varying position but with the chr 15 locus fixed at its estimated position, to the single-QTL model with just the chr 15 locus. The chr 15 curve is similar.

    These are actually slightly lower than the curves obtained from the single-QTL analysis with `scanone()`.

    ```
    out.hk <- scanone(sug, method="hk")
    plot(out.hk, chr=c(7,15), col="red", add=TRUE)
    ```

92. The function `addqtl()` is used to scan for an additional QTL to be added to the model.

    ```
    out.aq <- addqtl(sug, qtl=rqtl, method="hk")
    ```

    The biggest peaks are on chr 8 and 12, but nothing is particularly exciting.

    ```
    plot(out.aq)
    ```

    There is also a function `addpair()`, for scanning for a pair of QTL to be added.

93. Finally, we consider the function `stepwiseqtl()`, which is our fully automated stepwise algorithm to optimize the penalized LOD scores of Manichaikul et al. (2009). We first need to calculate the appropriate penalties from the two-dimensional permutation results.

    ```
    print(pen <- calc.penalties(operm2))
    ```

    We then run `stepwiseqtl()`, using `max.qtl=5`. It will perform forward selection to a model with 5 QTL, followed by backward elimination, and will report the model giving the largest penalized LOD score. The output is a QTL object.

    ```
    out.sq <- stepwiseqtl(sug, max.qtl=5, penalties=pen, method="hk", verbose=2)
    out.sq
    ```

    With `verbose=2`, we get an indication of the location of the QTL at each step.

    The result is exactly the model that we had after `refineqtl()`.

References

Broman KW, Sen Ś (2009) A guide to QTL mapping with R/qtl. Springer

Dalgaard P (2008) Introductory statistics with R, 2nd edition. Springer

Lincoln SE, Lander ES (1992). Systematic detection of errors in genetic linkage data. Genomics 14, 604–610

Manichaikul A, Moon JY, Sen Ś, Yandell BS, Broman KW (2009) A model selection approach for the identification of quantitative trait loci in experimental crosses, allowing epistasis. Genetics 181:1077–1086

Sugiyama F, Churchill GA, Li R, Libby LJM, Carver T, Yagami K-I, John SWM, Paigen B (2002) QTL associated with blood pressure, heart rate, and heart weight in CBA/CaJ and BALB/cJ mice. Physiol Genomics 10:5–12

Venables WN, Ripley BD (2002) Modern applied statistics with S, 4th edition. Springer