

Data checking

Our ability to map the loci contributing to variation in a trait depends critically on the quality and integrity of the data. Odd mapping results can often be traced to errors in the genotype data, the genetic maps, or the phenotype data. Thus, the first order of business, following data import, should be to identify and correct errors in the data.

A variety of data diagnostics are provided in R and R/qtl. We illustrate these below using real but anonymized data. The process of checking data can be quite interesting detective work. The features that should be studied are generally well characterized, but in many cases it can be tricky to identify the primary cause of a particular problem.

3.1 Phenotypes

We first take a look at the phenotype data. We look for individuals with unusual phenotypes. These may be truly unusual individuals, but they may also indicate errors in data entry and so deserve careful follow-up. We also look for systematic problems in the phenotype data (such as drifts in the measurements over time or between batches).

We begin by considering the example data, **ch3a**. We use the `library` function to load the `qtl` and `qtlbook` packages (the latter contains the example data used in this book), and use the `data` function to make the **ch3a** data set available to us.

```
> library(qtl)
> library(qtlbook)
> data(ch3a)
```

These data have five related phenotypes; Fig. 3.1 contains histograms of the phenotypes. Note that the histograms are generally skewed; this may influence our choice of QTL mapping method, or we may seek to transform

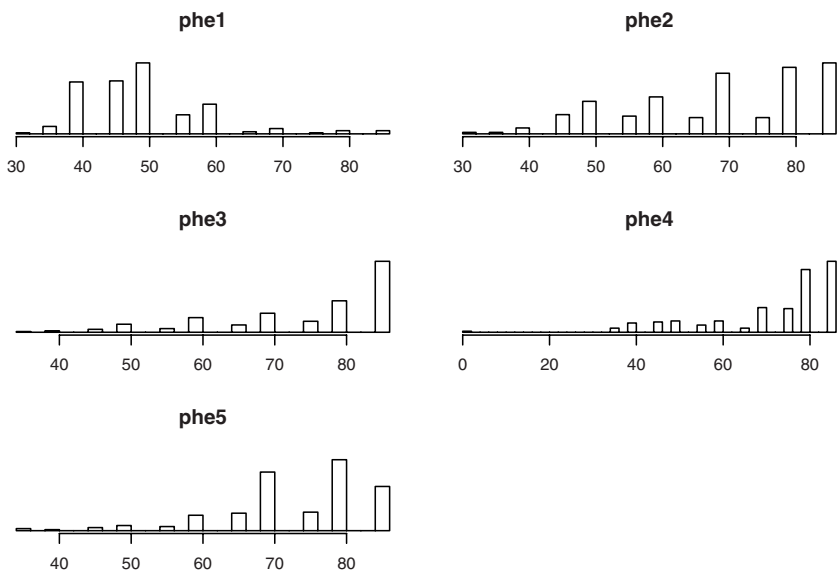


Figure 3.1. Histograms of the phenotypes from the `ch3a` data.

the phenotypes. More importantly, though, note that there is one individual whose fourth phenotype is 0, considerably lower than the other individuals.

Figure 3.1 may be produced with the following code.

```
> par(mfrow=c(3,2))
> for(i in 1:5)
+   plot.pheno(ch3a, pheno.col=i)
```

The function `par` is used to modify graphics parameters; we create three rows and two columns of plots with `mfrow=c(3,2)`. The function `c` is used to combine multiple items together into a vector.

We step through the five phenotypes using a `for` loop. (The `plot.pheno` function is called five times, with `i` taking the values 1, 2, ..., 5, sequentially.) The distribution of a phenotype may be displayed with the `plot.pheno` function, which will create either a histogram or a bar plot, according to whether the phenotype is numeric or categorical.

The unusual individual is rather difficult to see in Fig. 3.1; it is more clear in scatterplots of the phenotypes against one another, displayed in Fig. 3.2. Each panel contains the data for one phenotype plotted against the data for another phenotype. The individual with 0 at the fourth phenotype now stands out.

Figure 3.2 was created with the following code.

```
> pairs(jitter( as.matrix(ch3a$pheno) ), cex=0.6, las=1)
```

Since the phenotypes are discrete, we use `jitter` to add a bit of noise so that individual points may be distinguished. The code `ch3a$pheno` is used to

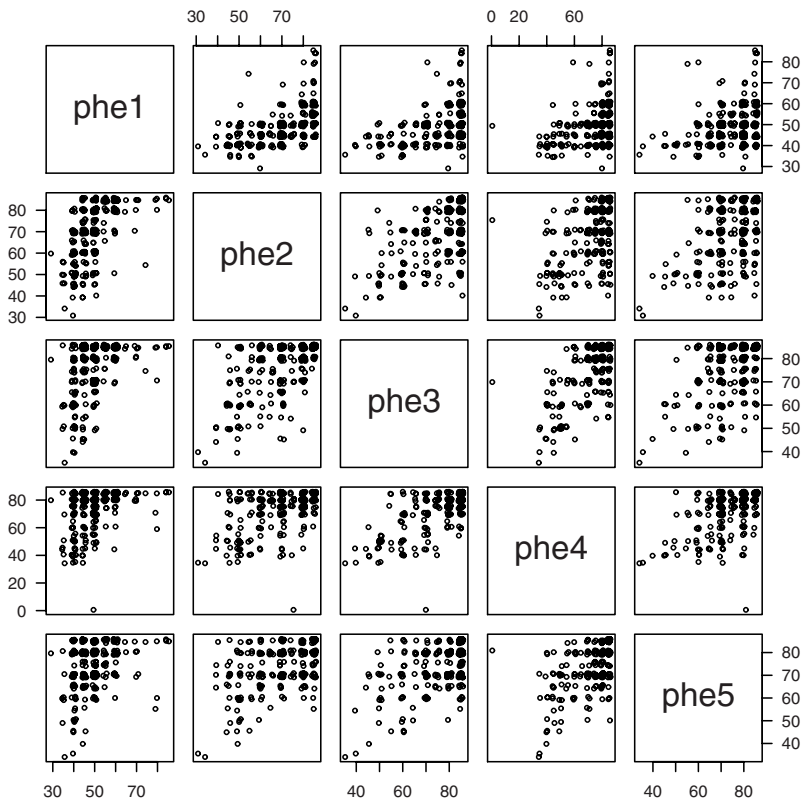


Figure 3.2. Scatterplots of the phenotypes against one another for the `ch3a` data.

pull out the phenotype data, which must be converted to a numeric matrix with `as.matrix` in order to use the `jitter` function. The function `pairs` creates the set of scatterplots; `cex=0.6` is used to change the size of the points and `las=1` is used to change the orientation of the y -axis labels.

It is best to go back to the primary data to determine whether the 0 is a true phenotype or whether it is a data entry error. In the latter case, we may set the phenotype to be missing as follows.

```
> ch3a$pheno[ch3a$pheno == 0] <- NA
```

Any 0 phenotypes will then be replaced with `NA` and therefore are treated as missing.

To complete our exploration of the phenotype data, we plot the individuals' phenotypes against their index, which may correspond to the order in which they were measured. The left panel in Fig. 3.3 contains a plot of the average of the five phenotypes for each individual, in the order they appeared in the

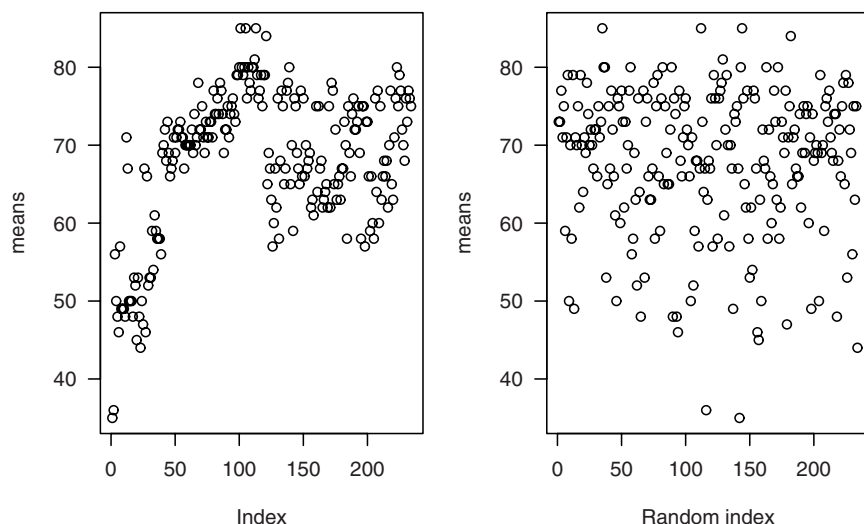


Figure 3.3. Plots of the average phenotype against index (left panel) and against a randomized index (right panel) for the **ch3a** data.

data. The right panel contains the same individual phenotype averages, but in a random order.

There is a clear pattern in the average phenotype that is not seen in the case that the data have been randomized (which we include just to emphasize the point). If this truly represents systematic changes in the phenotype over the course of the measurements, it is cause for considerable concern, as it indicates an important source of uncontrolled (and nongenetic) variation.

Figure 3.3 was produced with the following code. The function `apply` is used to obtain the row averages in the phenotype data, and `sample` is used to randomize the order of these averages.

```
> par(mfrow=c(1,2), las=1, cex=0.8)
> means <- apply(ch3a$pheno, 1, mean)
> plot(means)
> plot(sample(means), xlab="Random index", ylab="means")
```

3.2 Segregation distortion

It is important to check for segregation distortion at all markers (i.e., that the genotypes appear in the expected proportions), as apparent distortion may indicate genotyping problems. For example, consider the example data, **ch3b**. We use the function `geno.table` to inspect the genotype frequencies at each marker. The last column in the output is a p -value for a χ^2 test of Mendelian proportions (1:2:1 in an intercross).

```
> data(ch3b)
> gt <- geno.table(ch3b)
> gt[ gt$P.value < 1e-7, ]
```

	chr	missing	AA	AB	BB	not.BB	not.AA	AY	BY	P.value
c4m7	4	0	31	0	113	0	0	0	0	2.829e-52
c6m9	6	0	28	0	116	0	0	0	0	2.374e-55
c7m3	7	40	62	2	40	0	0	0	0	1.257e-23
c10m8	10	64	36	4	40	0	0	0	0	6.950e-15
c11m3	11	26	10	0	108	0	0	0	0	1.070e-61
c13m1	13	18	99	27	0	0	0	0	0	1.923e-43
c13m4	13	51	43	13	37	0	0	0	0	2.241e-11
c14m1	14	57	10	77	0	0	0	0	0	1.979e-12
c14m2	14	0	61	79	4	0	0	0	0	8.048e-11
c14m5	14	18	45	1	80	0	0	0	0	1.900e-31
c16m2	16	12	0	104	28	0	0	0	0	8.293e-13
c16m6	16	15	32	1	96	0	0	0	0	1.149e-41
c18m5	18	38	1	1	104	0	0	0	0	2.379e-66
c19m3	19	9	0	134	1	0	0	0	0	3.500e-29

We see 14 markers, on 11 chromosomes, showing quite extreme distortion. For example, markers c4m7 and c6m9 had no heterozygotes, while marker c19m3 had almost all heterozygotes. It is likely that these problems are due to genotyping errors.

As a further example, let us look at the *listeria* data from Boyartchuk *et al.* (2001), which is included with R/qtl. Some markers show segregation distortion, but the distortion is much less severe than was observed in the ch3b data.

```
> data(listeria)
> gt <- geno.table(listeria)
> p <- gt$P.value
> gt[ !is.na(p) & p < 0.01, ]
```

	chr	missing	CC	CB	BB	not.BB	not.CC	P.value
D6M284	6	0	27	76	17	0	0	0.0060967
D6M254	6	0	18	77	25	0	0	0.0053804
D12M46	12	33	34	33	20	0	0	0.0083345
D13M99	13	0	48	49	23	0	0	0.0007282
D13M233	13	14	41	43	22	0	0	0.0050294
D13M106	13	0	45	55	20	0	0	0.0036066
D13M147	13	0	45	55	20	0	0	0.0036066

Note that `is.na` returns TRUE or FALSE according to whether the values are missing or not, respectively, and `!` is the logical “not.”

Many markers on chromosome 13 show a reduced frequency of B alleles, which may indicate real segregation distortion (e.g., that there is a locus

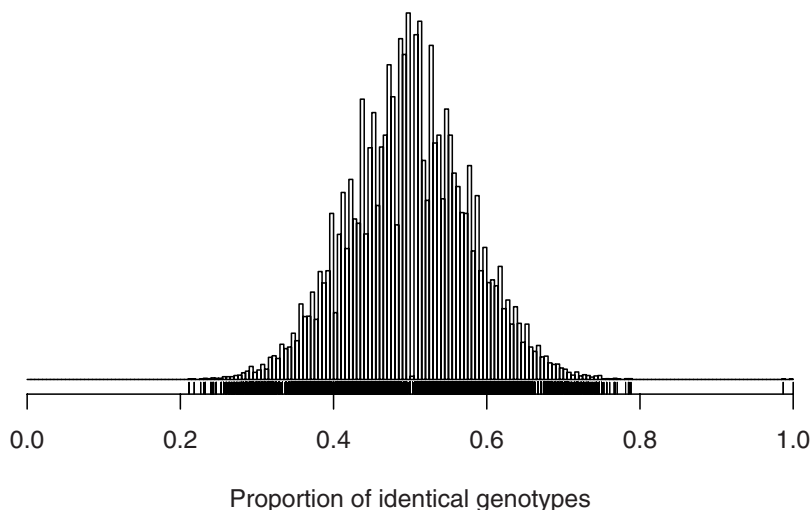


Figure 3.4. Histogram of the proportion of markers with identical genotypes for each pair of individuals in the `ch3a` data.

on that chromosome that is associated with early mortality), rather than genotyping errors.

3.3 Compare individuals' genotypes

We have occasionally found it useful to compare the genotype data for each pair of individuals from a cross, to identify pairs that have unusually similar genotypes. These may indicate sample mix-ups of some kind.

For example, Fig. 3.4 contains a histogram of the proportion of markers with identical genotypes for each pair of individuals in the `ch3a` data. There are two pairs of individuals that have very similar genotype data.

Figure 3.4 was created with the following code. The `comparegeno` function returns a matrix whose (i, j) th element is the proportion of markers at which individuals i and j have the same genotype. The function `hist` creates the actual histogram; the argument `breaks` defines the number of bins (and can be used to define the precise breakpoints for the bins). The function `rug` is used to create, underneath the histogram, line segments at the individual data points, so that the two outliers may be clearly seen.

```
> data(ch3a)
> cg <- comparegeno(ch3a)
> hist(cg, breaks=200,
+       xlab="Proportion of identical genotypes")
> rug(cg)
```

With the following code, we can identify the pairs of individuals with very similar genotype data.

```
> which(cg > 0.9, arr.ind=TRUE)
```

```
      row col
[1,] 138   5
[2,]  55  12
[3,]  12  55
[4,]   5 138
```

Individuals 5 and 138 have identical genotypes at all 86 markers at which they were both typed; individuals 12 and 55 have the same genotype at 75/76 markers. Real backcross individuals shouldn't show such similarity in their genotypes, and so these individuals' data should be viewed with suspicion.

3.4 Check marker order

It is critical that one check that markers are placed on the correct chromosomes and in the correct order, as the incorrect placement of markers on the map can destroy the results of the QTL analysis. Even if marker positions are based on a high-quality physical map, mislabeling of markers can occur, and so marker labels may not match the true markers that were genotyped.

3.4.1 Pairwise recombination fractions

The first thing to do is to estimate, for each pair of markers, the recombination fraction between them, r , and calculate a LOD score for the test of $r = 1/2$. Markers on different chromosomes should not appear linked, and for markers on the same chromosome, the estimated recombination fraction should be smaller for more closely linked markers.

We again consider a set of real but anonymized data, included in the `qtlbook` package. We use `est.rf` to estimate the recombination fractions between all pairs of markers. It inserts the results back into the cross object, and so we assign the results back to the object, `ch3c`.

```
> data(ch3c)
> ch3c <- est.rf(ch3c)
```

Warning message:

```
In est.rf(ch3c) : Alleles potentially switched at markers
      c1m3 c1m4 c7m1 c7m2
```

Before proceeding further, note the warning message produced by `est.rf`. There may be markers whose alleles got switched (A for B and B for A). These potential problems are identified by looking for markers whose LOD

scores are larger for cases with $\hat{r} > 0.5$ rather than $\hat{r} < 0.5$. The function `checkAlleles` gives slightly more detail; the last column in the output is the difference between the largest LOD score corresponding to an estimated recombination fraction > 0.5 and the largest LOD score corresponding to an estimated recombination fraction < 0.5 . Note that markers that are tightly linked to a problem marker will also show up in this table.

```
> checkAlleles(ch3c)

  marker chr index diff.in.max.LOD
2   c1m3   1     2         21.378
3   c1m4   1     3         15.010
32  c7m1   7     1          6.691
33  c7m2   7     2         10.769
```

There appear to be problems on chromosomes 1 and 7. Let us look in more detail at the genotype data for the markers on chr 1. We use `pull.map` to display the map for that chromosome, so that we can see the other marker names, and then use `geno.crosstab` to create tables of genotypes at one marker against genotypes at another marker.

```
> pull.map(ch3c, 1)

c1m1 c1m3 c1m4 c1m5
  8.3 49.0 59.5 89.0

> geno.crosstab(ch3c, "c1m3", "c1m4")
```

```
      c1m4
c1m3 - AA AB BB
-    7  0  0  0
AA   0  0  3 19
AB   0  0 38  7
BB   0 22  3  1
```

```
> geno.crosstab(ch3c, "c1m3", "c1m5")
```

```
      c1m5
c1m3 - AA AB BB
-    7  0  0  0
AA   0  2 11  9
AB   0  9 24 12
BB   0 12 12  2
```

```
> geno.crosstab(ch3c, "c1m4", "c1m5")
```

```
      c1m5
c1m4 - AA AB BB
-    7  0  0  0
```

```
AA 0 11 10 1
AB 0 11 28 5
BB 0 1 9 17
```

It looks like marker 2 ("c1m3") is the problem: for that marker, relative to markers c1m4 and c1m5, the double-recombinant classes are more common than the nonrecombinant ones, while the table of two-locus genotypes for markers c1m5 and c1m5 looks okay.

To fix the problem, we pull out the genotypes for chromosome 1 using the function `pull.geno`, swap the alleles (replacing 1's with 3's and vice versa), and then put the new data back.

```
> g <- pull.geno(ch3c, 1)
> g[, "c1m3"] <- 4 - g[, "c1m3"]
> ch3c$geno[[1]]$data <- g
```

By a similar approach, we find that it is marker 2 ("c7m2") on that is the problem one on chr 7. We fix it as follows.

```
> g <- pull.geno(ch3c, chr=7)
> g[, "c7m2"] <- 4 - g[, "c7m2"]
> ch3c$geno[[7]]$data <- g
```

If we now rerun `est.rf` and `checkAlleles`, we'll find there are no further problems of this form.

```
> ch3c <- est.rf(ch3c)
> checkAlleles(ch3c)
```

No apparent problems.

We now return to the recombination fractions themselves, and our assessment of marker placement. The function `plot.rf` is used to plot the pairwise recombination fractions and LOD scores. We use the option `alternate.chrid=TRUE` so that the individual chromosome IDs may be more easily distinguished.

```
> plot.rf(ch3c, alternate.chrid=TRUE)
```

The results are displayed in Fig. 3.5; the estimated recombination fractions between markers are in the upper left, and the LOD scores are in the lower right. Red indicates pairs of markers that appear to be linked (low $\hat{r}$ or high LOD), and blue indicates pairs that are not linked (high $\hat{r}$ or low LOD).

There are a number of red points in the lower right, indicating markers on different chromosomes that appear linked. In particular, there appear to be problems on chromosomes 1, 7, 12, 13, and 15. With the following code, we plot the results for just those chromosomes (see Fig. 3.6).

```
> plot.rf(ch3c, chr=c(1, 7, 12, 13, 15))
```

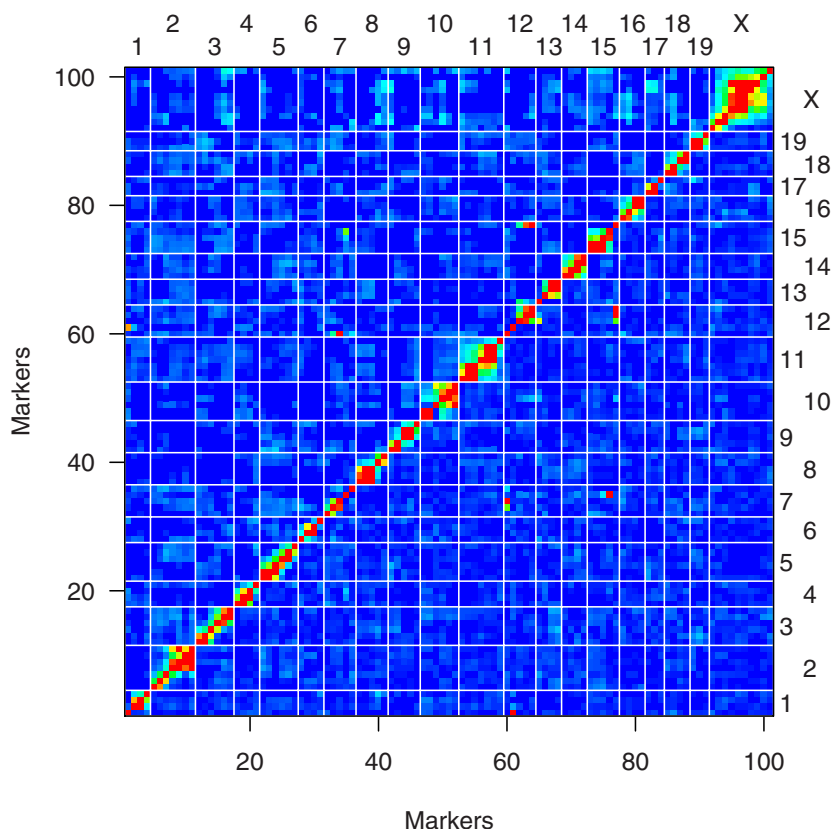


Figure 3.5. Estimated recombination fractions (upper left) and LOD scores (lower right) for all pairs of markers in the *ch3c* data.

As a further indication of this problem, it is valuable to use the available genotype data to reestimate the intermarker distances of the genetic map. This is done, and the map plotted, with the following code. Note that the `nm` object will have class `"map"`, and so `plot(nm)` is equivalent to `plot.map(nm)`. Also note that with `error.prob=0.001`, we assume a 0.1% genotyping error rate.

```
> nm <- est.map(ch3c, error.prob=0.001)
> plot(nm)
```

The estimated map, shown in Fig. 3.7, indicates clear problems on chromosomes 7, 12 and 15: enormous map expansion occurs as a result of markers that do not belong on those chromosomes. There may also be problems on chromosomes 10 and 13. The results in Fig. 3.6 indicate that the fourth marker on chr 7 belongs on chr 15, the first marker on chr 12 belongs on chr 7, the

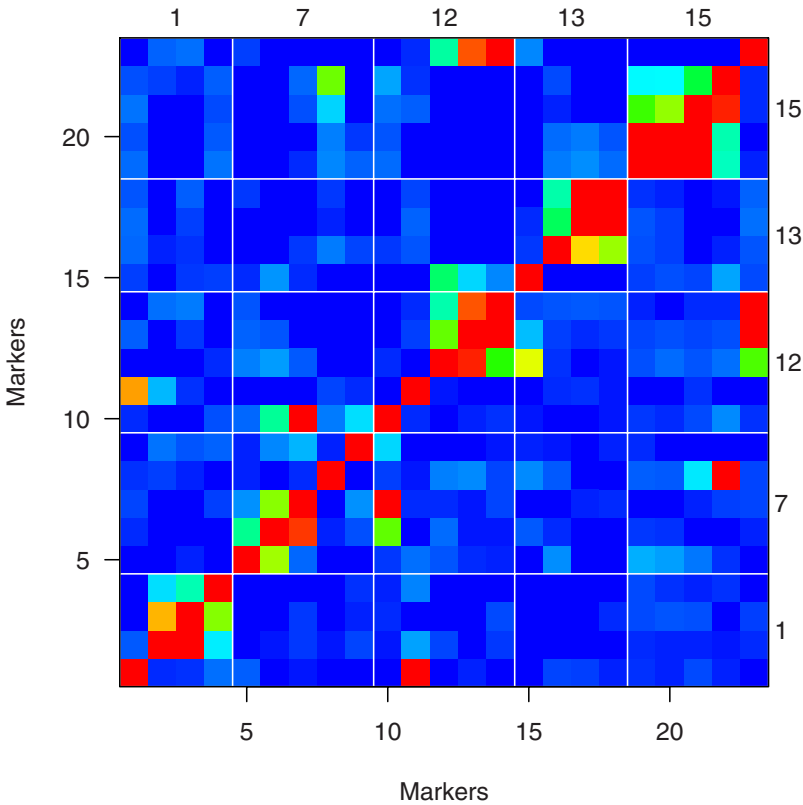


Figure 3.6. Estimated recombination fractions (upper left) and LOD scores (lower right) for pairs of markers on selected chromosomes in the `ch3c` data.

second marker on chr 12 belongs on chr 1, the first marker on chr 13 belongs on chr 12, and the fifth marker on chr 15 belongs on chr 12.

It is questionable whether we should move these markers to the positions that they appear to be linked, or just omit them. The ideal solution would be to retype these markers starting with new material. If we want to use the available data for an initial analysis, and so seek to fix these problems in marker positions, `R/qtl` does have some limited facilities for moving markers between chromosomes.

We first need to identify the names of the markers, which can be accomplished with the function `find.marker`, using the argument `index` to specify the markers by their numeric indices within the chromosomes. We then use the `movemarkers` function to move the markers to the chromosomes that they appear to belong. (The markers are moved to the end of the chromosome, and so we will later need to fix the marker order on those chromosomes.)

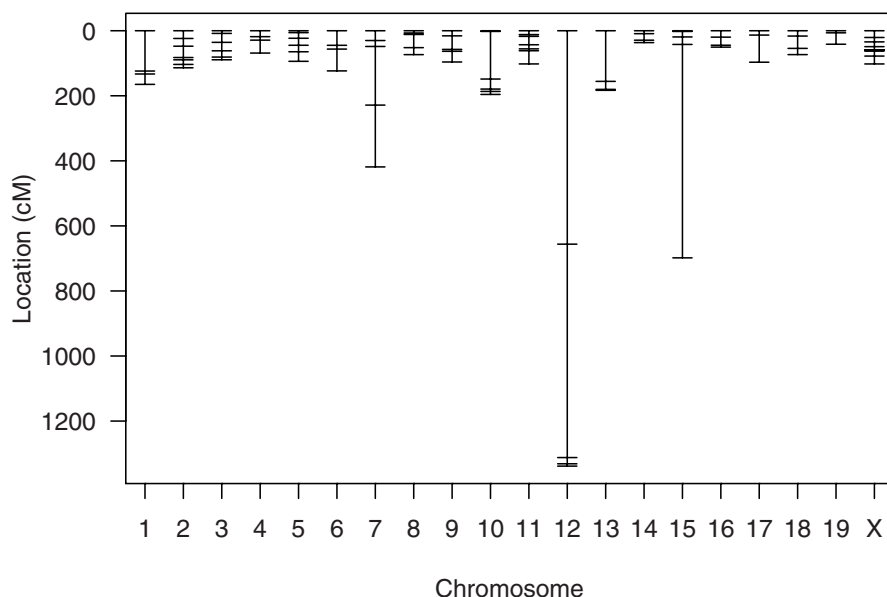


Figure 3.7. Genetic map, as estimated from the `ch3c` data.

```
> ch3c <- movemarker(ch3c, find.marker(ch3c, 7, index=4), 15)
> ch3c <- movemarker(ch3c, find.marker(ch3c, 12, index=2), 1)
> ch3c <- movemarker(ch3c, find.marker(ch3c, 12, index=1), 7)
> ch3c <- movemarker(ch3c, find.marker(ch3c, 13, index=1), 12)
> ch3c <- movemarker(ch3c, find.marker(ch3c, 15, index=5), 12)
```

We needed to be careful to move the second marker on chr 12 before moving the first marker; if we had first moved the initial marker, the second marker would no longer be in the second position.

We can then use `est.rf` to plot the recombination fractions and LOD scores for the relevant chromosomes again. The results are in Fig. 3.8. The markers now appear to be on the correct chromosomes, though there remain some problems with the order of markers within the chromosomes. (We will address that issue in Sec. 3.4.2.)

One last point on pairwise linkages: some strategies for selective genotyping can lead to odd results in the pairwise recombination fractions. For example, consider the `hyper` data. At most markers, only individuals with extreme phenotypes were genotyped, and at some markers, only individuals showing recombination events at the surrounding markers were genotyped. (The pattern of missing genotype data is displayed in Fig. 1.7 on page 10). The latter strategy leads to somewhat odd results in the pairwise recombination fractions and LOD scores, as in estimating the recombination fraction between a pair of markers, we consider only the data on individuals who were typed at both markers.

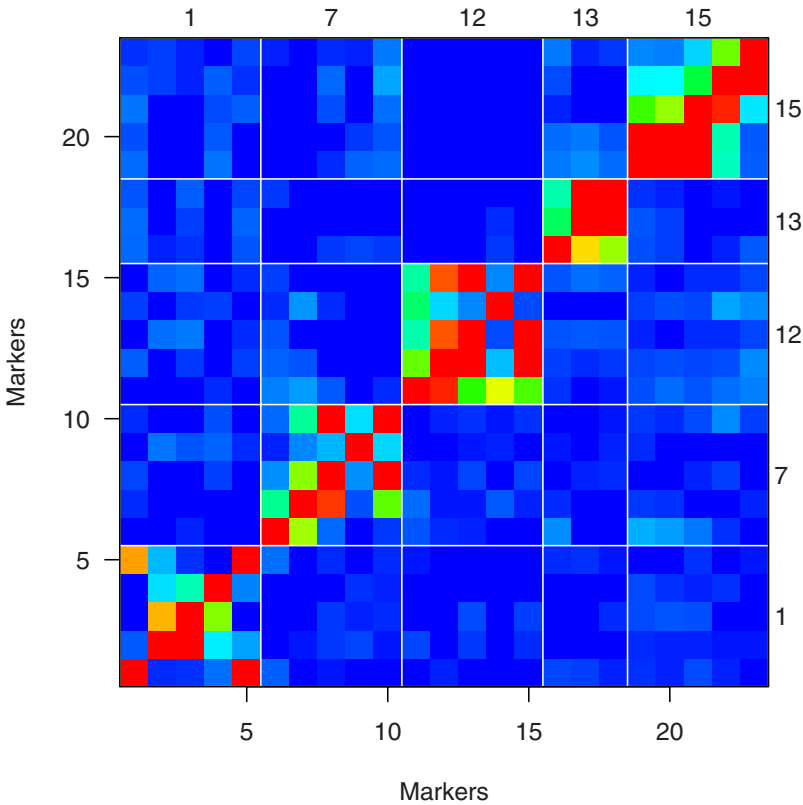


Figure 3.8. Estimated recombination fractions (upper left) and LOD scores (lower right) for pairs of markers on selected chromosomes in the `ch3c` data, after some problems with marker positions have been fixed.

To calculate the recombination fractions for the `hyper` data set, we do the following. The results are displayed in Fig. 3.9.

```
> data(hyper)
> hyper <- est.rf(hyper)
> plot.rf(hyper, alternate.chrid=TRUE)
```

Note that, while the LOD scores in the lower right triangle indicate no linkages between nonsyntenic markers, there are some pairs with low estimated recombination fractions (red pixels in the upper left triangle), as some markers were typed on very few individuals. The checkerboard patterns on chr 1, 4, 11, and 15, might indicate problems with marker order, but are really due to the strategy of typing only recombinant individuals at some markers.

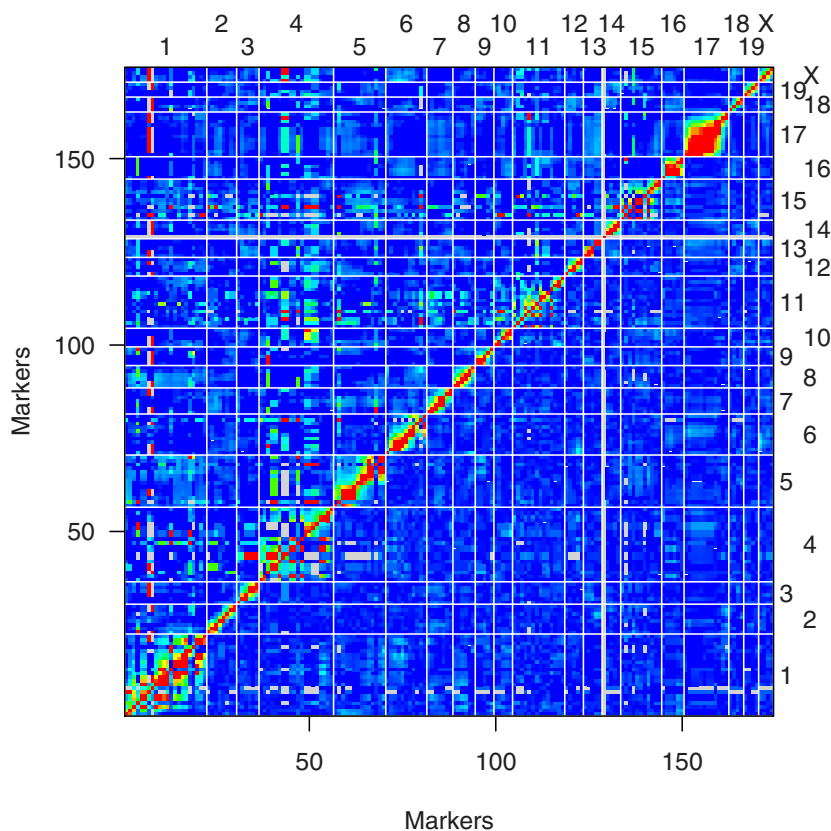


Figure 3.9. Estimated recombination fractions (upper left) and LOD scores (lower right) for all pairs of markers in the **hyper** data.

3.4.2 Rippling marker order

One can check the order of markers on a chromosome using the **ripple** function (whose name was taken from a similar function in MapMaker/EXP).

We would like to compare all possible orderings of markers on a chromosome, but with even a moderate number of markers, the number of possible marker orders is too large for such an exhaustive evaluation. If there are n markers on a chromosome, there are $n!/2$ possible marker orders, where $n! = n \times (n - 1) \times (n - 2) \times \cdots \times 2 \times 1$. In the case of 10 markers, there are 1,814,400 possible marker orders. Thus, in **ripple** we consider a sliding window of markers and consider all possible orders of the markers within the window, keeping the order of markers outside the window fixed.

The **ripple** function can use two methods for comparing orders: maximum likelihood and minimal obligate crossovers. Maximum likelihood (in which one considers the probability of the observed genotype data given a

particular order, and then chooses the marker order for which this probability is maximized) is generally preferred, but is considerably more computationally intensive. It is simpler to count the number of obligate crossovers in the data, for a given order, and then choose the order for which the number of obligate crossovers is minimized. (In a backcross, one is simply counting crossovers, though with the assumption that no double crossovers occur between typed markers.) Counting crossovers is hundreds of times faster than maximum likelihood, and the results are remarkably similar. Thus we generally recommend using the crossover count method with a large window, followed by maximum likelihood with a much smaller window.

The key arguments for the `ripple` function include the `cross`, the specified chromosome (`chr`; just one chromosome is considered at a time), the `window` size, and the `method` (either `"countxo"` or `"likelihood"`). For the likelihood method, one may also specify an assumed genotyping error rate with `error.prob`.

Let us return to the `ch3c` data; we had ensured that all markers were on the correct chromosomes, but saw that some chromosomes showed clear problems in marker order. We first look at chromosome 1, for which there are just five markers. We'll first count crossovers, and look at all possible marker orders. We do so as follows.

```
> rip <- ripple(ch3c, 1, 5)
```

```
60 total orders
```

The result (assigned to the object `rip`) contains the number of obligate crossovers for each possible marker order. We can get a summary of the results as follows.

```
> summary(rip)
```

						obligX0					
Initial	1	2	3	4	5	197					
1		1	5	2	3	4	124				
2			4	3	2	1	5	134			
3				2	3	4	1	5	146		
4					1	5	4	3	2	146	
5						1	5	3	2	4	152

... [16 additional rows] ...

The first row is the original marker order (i.e., that which is in the data). Other marker orders are sorted by the number of obligate crossovers (which appears in the final column), but only the first few are displayed. Note that by moving marker 5 from one side of the chromosome to the position between markers 1 and 2, the number of obligate crossovers is reduced from 197 to 124. (Marker 5 was originally on chromosome 12, and when we used `movemarkers` to move it to chromosome 1, we just placed it at the end.)

We can adopt the second order (with the minimal number of obligate crossovers) using the `switch.order` function, whose main arguments are `cross`, `chr`, and `order`. The `order` object should be a vector of integers indicating the new marker order; we may insert the second row from the output of `ripple` directly, to save a bit of effort, even though it is one value longer than the number of markers. The `switch.order` function also takes an argument `error.prob`: the assumed genotyping error rate in the estimation of the genetic map for the new order. Thus, we can switch the marker order on chromosome 1 with the following.

```
> ch3c <- switch.order(ch3c, 1, rip[2,])
```

With the markers in their new order, we will now run `ripple` again using the likelihood method but with a smaller window size, to see if the likelihood approach is inconsistent with the approximate method. We assume a genotyping error rate of 0.001.

```
> rip <- ripple(ch3c, 1, 3, method="likelihood",
+             error.prob=0.001, verbose=FALSE)
> summary(rip)
```

		LOD	chr	len
Initial	1 2 3 4 5	0.0	117.3	
1	2 1 3 4 5	-1.2	173.7	

When one uses `method="likelihood"`, the `ripple` function gives some indication of its progress; we suppress this by using `verbose=FALSE`.

The second-to-last column in the summary is a LOD score ($\log_{10}$ likelihood ratio) comparing the original order to the alternative order (or orders); a negative value (as here) indicates that the original order has higher likelihood. The last column gives the estimated genetic length of the chromosome with the different marker orders; the best marker order is generally that giving the shortest chromosome length. We see that no further change is needed.

We would now use the same approach with all other chromosomes. While one could type the above commands repeatedly, a more detailed knowledge of R can save quite a bit of effort. For example, we can use a `for` loop to run `ripple` for each chromosome, one at a time, as follows. (We include chromosome 1 again, to make the code simpler.)

```
> rip <- vector("list", nchr(ch3c))
> names(rip) <- names(ch3c$geno)
> for(i in names(ch3c$geno))
+   rip[[i]] <- ripple(ch3c, i, 7, verbose=FALSE)
```

The first line creates a “list” that will contain the output. The second line assigns it the names of the chromosomes in the data.

Now things get a bit hairy. We use `sapply` to pull out, for each chromosome, the difference in the number of obligate crossovers between the initial order and the best of the other orders.

```
> dif.nxo <- sapply(rip, function(a) a[1,ncol(a)]-a[2,ncol(a)])
> dif.nxo
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-10	16	-7	-23	-8	-22	44	-10	-12	45	-12	78	-5	-10	-4
16	17	18	19	X										
-10	-11	-5	-9	2										

It is probably best to not seek a complete understanding of this code at the moment. We assigned the results to the object `dif.nxo`, and then typed the name of the object to print the results. The positive numbers indicate that an order other than that in the data showed a decrease in the number of obligate crossovers.

We can now loop through all of the chromosomes and switch the order of markers whenever the alternate order showed an improvement.

```
> for(i in names(ch3c$geno)) {
+   if(dif.nxo[i] > 0)
+     ch3c <- switch.order(ch3c, i, rip[[i]][2,])
+ }
```

Again, the code is complex, but consider the effort saved; we hope this encourages the reader to learn a bit more R.

Since for some chromosomes, we did not look at all possible marker orders, it is a good idea to repeat the process to see if any further improvement may be found.

```
> for(i in names(ch3c$geno))
+   rip[[i]] <- ripple(ch3c, i, 7, verbose=FALSE)
> dif.nxo <- sapply(rip, function(a) a[1,ncol(a)]-a[2,ncol(a)])
```

We can now look to see whether any of the values in `dif.nxo` are positive (indicating that an alternate order is better).

```
> any(dif.nxo > 0)

[1] FALSE
```

Finally, we go back through all of the chromosomes with `ripple`, this time using `method="likelihood"` and a window size of three markers.

```
> for(i in names(ch3c$geno))
+   rip[[i]] <- ripple(ch3c, i, 3, method="likelihood",
+                     error.prob=0.001, verbose=FALSE)
> lod <- sapply(rip, function(a) a[2, ncol(a)-1])
```

The object `lod` contains the LOD scores for the best alternate order relative to that in the data. The following prints those that are positive (indicating that the alternate order is an improvement).

```
> lod[lod > 0]
```

X
1.655

The X chromosome shows some improvement, and so we look at those results more closely.

```
> summary(rip[["X"]])
```

												LOD	chr	len
Initial	1	2	3	4	5	6	7	8	9	10		0.0	96.9	
1	1	2	3	5	4	6	7	8	9	10		1.7	102.2	
2	1	2	3	5	6	4	7	8	9	10		1.7	102.2	
3	1	2	3	4	5	7	6	8	9	10		0.0	96.9	

Switching markers 4 and 5 increases the likelihood by a factor of $10^{1.7} \approx 50$, but leads to a longer chromosome. It is questionable whether the marker order should be switched or not, as a LOD score of 1.7 is not exceptionally strong evidence for the alternate order. Both orders might be considered in later QTL analyses, and if there is evidence for a QTL in the region, we will want to look especially carefully at the marker order.

Finally, we may take another look at the pairwise recombination fractions, at least for chromosomes that were shown in Fig. 3.8 to have some problems. Calls to `est.rf` and `plot.rf` produce the results in Fig. 3.10, as follows.

```
> ch3c <- est.rf(ch3c)
> plot.rf(ch3c, chr=c(1, 7, 12, 13, 15))
```

The results are just what we want: red along the diagonal, fading to blue off the diagonal.

3.4.3 Estimate genetic map

Now that we have the markers in what we believe are the correct orders, we finish this section by estimating the intermarker distances from the observed data; we further compare the results to the map that was included with the data. The function `est.map` does the map estimation. We can use `plot.map` to plot a single genetic map or to plot two maps against each other. Moreover, if a cross object is input to this function, it pulls out the genetic map from the data and plots the map. So with the following, we can estimate the genetic map for the `ch3c` data in its final form and plot it against the map in the data (see Fig. 3.11).

```
> nm <- est.map(ch3c, error.prob=0.001, verbose=FALSE)
> plot.map(ch3c, nm)
```

For many chromosomes, the estimated map is identical to the one within the **ch3c** data set. That is because we had moved some markers around, and if marker order is modified, the intermarker distances must be estimated with

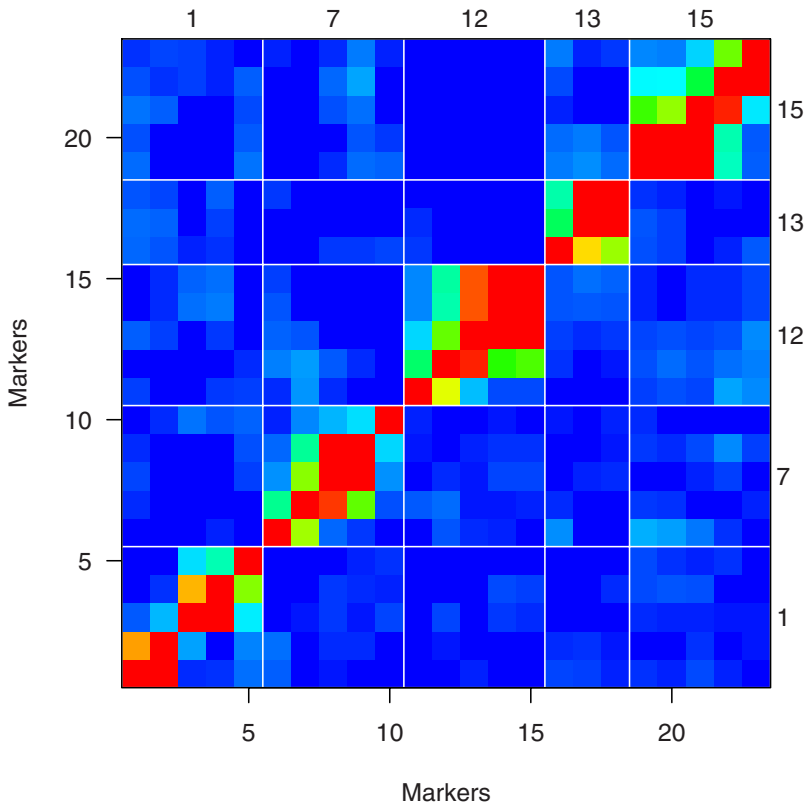


Figure 3.10. Estimated recombination fractions (upper left) and LOD scores (lower right) for pairs of markers on selected chromosomes in the `ch3c` data, after some problems with marker positions have been fixed.

the available data. Several chromosomes exhibit considerable map expansion (e.g., chromosome 6): the estimated map is quite a bit longer than the map in the data. This may indicate the presence of genotyping errors.

One may wish, at this point, to replace the map within the `ch3c` with that estimated from the data. Reference genetic maps are often based on a rather small number of individuals. (For example, the original MIT mouse genetic map was based on an intercross with just 46 individuals.) One's own data often contains many more individuals, and so may produce a more accurate map. The only caveat is that reference genetic maps generally contain a much more dense set of markers, which (as described in the next section) provides greater ability to detect genotyping errors. Thus reference genetic maps may be based on cleaner genotype data.

To replace the genetic map in the `ch3c` data with that estimated from the data, we use the function `replace.map`, as follows.

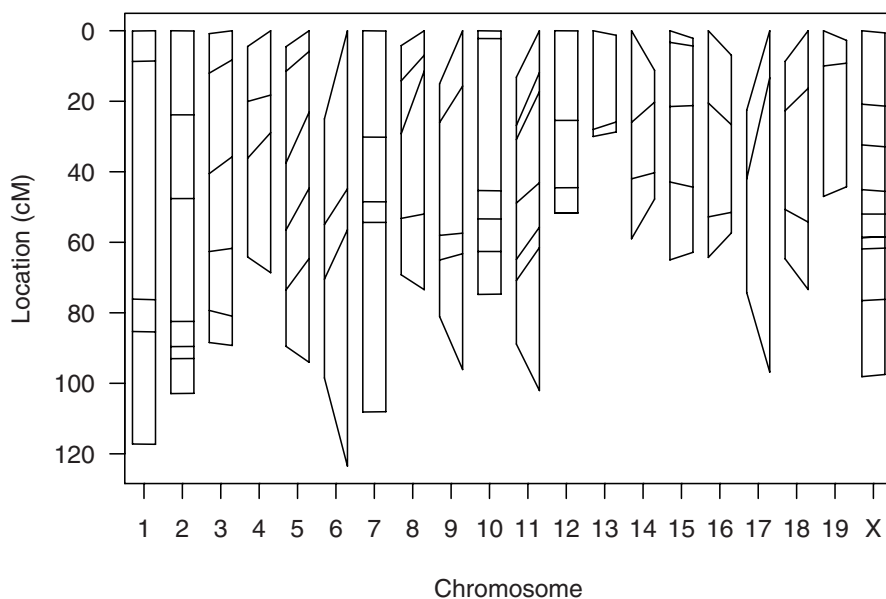


Figure 3.11. The genetic map in the *ch3c* data (after considerable revisions in marker order) plotted against the map estimated from the data. For each chromosome, the line on the left is the map in the data, and the line on the right is the map estimated from the data; line segments connect the positions for each marker.

```
> ch3c <- replace.map(ch3c, nm)
```

3.5 Identifying genotyping errors

Our ability to map QTL relies on high-quality genotype data; errors in the genotype data will lessen our ability to detect QTL. Genotyping errors may appear as apparent tight double crossovers. Meiosis generally exhibits strong crossover interference, and so crossovers will not occur too close together. Thus, if the genotype at a single marker is out of phase with the surrounding markers, it is likely in error. This requires dense marker genotype data; with sparse markers, one cannot be sure whether the apparent tight double crossover is an error or is a true double crossover.

Detection of genotyping errors is facilitated by the calculation of genotyping error LOD scores. For each individual at each marker, we calculate a $\log_{10}$ likelihood ratio comparing the hypothesis that the particular genotype is in error to the hypothesis that it is correct; the likelihood uses the genotype data at all other markers on the chromosome. (For further details on this calculation, see Sec. D.7.) These LOD scores serve largely to ease the identification of unusually tight double crossovers.

To calculate the genotyping error LOD scores, we use the `calc.errorlod` function. One must specify an assumed genotyping error rate, and the results can be sensitive to this rate. The results are especially sensitive to the genetic map. Thus, before calculating the error LOD scores, we may first wish to replace the map in the data with that estimated from the data. In the following we do that plus calculate the error LOD scores for the `hyper` data.

```
> data(hyper)
> newmap <- est.map(hyper, error.prob=0.01)
> hyper <- replace.map(hyper, newmap)
> hyper <- calc.errorlod(hyper)
```

The `top.errorlod` function prints information about the genotypes with error LOD score above a specified cut off (indicated by the argument `cutoff`). An argument `chr` may be used to give results for a selected subset of the chromosomes. In the following, we look at the genotypes with error LOD scores > 5 . We save the results to the object `top`, and then type the name of the object to print the results.

```
> top <- top.errorlod(hyper, cutoff=5)
> top
```

	chr	id	marker	errorlod
1	16	50	D16Mit171	16.000
2	16	54	D16Mit171	16.000
3	16	81	D16Mit5	8.915
4	16	24	D16Mit5	8.915
5	16	71	D16Mit5	8.915
6	16	34	D16Mit5	8.915
7	13	42	D13Mit78	8.000
8	13	42	D13Mit148	7.881

There are a number of genotypes indicated to be likely in error. The `id` column is a numeric index here, but if a phenotype named “`id`” or “`ID`” had been included in the data, such labels would be used.

We can look more closely at the problem genotypes with the function `plot.geno`, which plots the genotype data for a single chromosome, for selected individuals. We pull out the individual IDs from the result of `top.errorlod`, and plot their genotype data for chromosome 16. We use the `cutoff` argument to indicate that genotypes with error LOD score > 5 should be flagged.

```
> plot.geno(hyper, 16, top$id[top$chr==16], cutoff=5)
```

The results are shown in Fig. 3.12.

A small number of genotyping errors will not have much influence on the results, and so one should be concerned only if an inordinate number of possible errors are seen (though this may also indicate a problem with marker

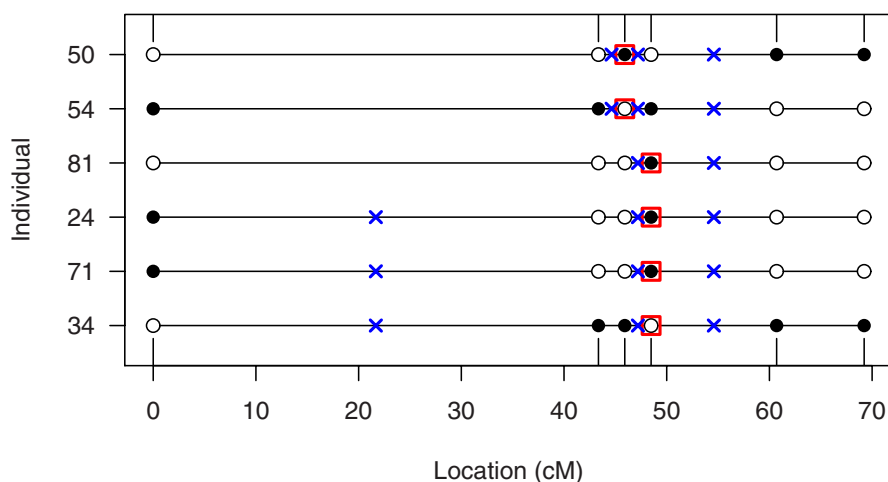


Figure 3.12. Chromosome 16 genotypes for selected individuals in the `hyper` data. Open and closed circles are homozygous and heterozygous genotypes, respectively. Possible genotyping errors are flagged with red squares; inferred crossovers are indicated with blue $\times$'s.

order). However, if one sees evidence for a QTL in the region, it may be valuable to take a second look at the raw genotype information or to rerun the genotypes on some markers.

3.6 Counting crossovers

Another useful diagnostic is to count the number of crossovers implied by the genotype data in each individual. Individuals with an unusually small or large number of crossovers should be viewed with suspicion. This may be an indication of either poor quality DNA or sample mix-ups.

The function `countX0` may be used to count the number of observed crossovers for each individual. One may use the `chr` argument to focus on a selected set of chromosomes. By default, we count the total number of crossovers across the genome. If `countX0` is called with the argument `by-chr=TRUE`, the function returns a matrix containing the numbers of crossovers on the individual chromosomes.

Let us again consider the `hyper` data. We may count crossovers and plot them as follows (see Fig. 3.13).

```
> nxo <- countX0(hyper)
> plot(nxo, ylab="No. crossovers")
```

Note that these counts are the minimal number of crossovers required to explain the observed genotype data. We see a large shift in the distribution

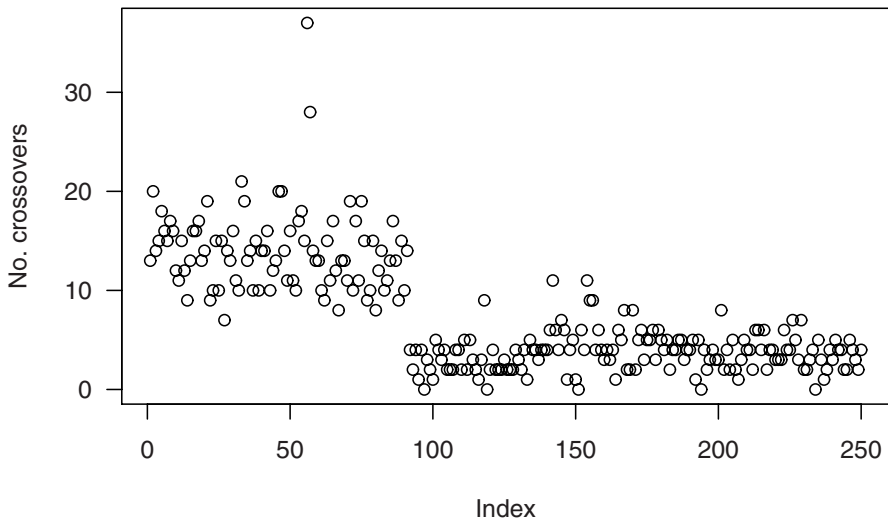


Figure 3.13. The observed number of crossovers for each individual in the *hyper* data.

between the first 92 individuals and the remaining 158 individuals, due to the selective genotyping of the data. (The initial 92 individuals were genotyped at markers across the genome; the remaining individuals were typed only on selected chromosomes that exhibited evidence for a QTL.)

Particularly interesting are the two individuals with > 25 crossovers.

```
> nxo[nxo>25]
```

```
56 57
```

```
37 28
```

The 56th individual exhibited 37 crossovers. (The first row in the above output contains labels with the indexes of the individuals; the second row contains the crossover counts.) This is considerably larger than was seen in others (see Fig. 3.13). The initial 92 individuals showed an average of 14 crossovers. The remaining 158 individuals (genotyped only on selected chromosomes) had an average of 4 crossovers.

```
> mean(nxo[1:92])
```

```
[1] 13.84
```

```
> mean(nxo[-(1:92)])
```

```
[1] 3.741
```

If we pull out the crossover counts for each chromosome for individual 56, we can identify the chromosomes that are particularly problematic.

```
> countX0(hyper, bychr=TRUE)[56,]
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19  X
2  0  1  1  5  7  2  2  1  2  4  1  0  1  4  1  1  1  1  0
```

The genotype data for chromosome 6 (for which this individual shows seven apparent crossovers) are particularly suspicious, and deserve further investigation.

3.7 Missing genotype information

As a final diagnostic, we compute the proportion of missing genotype information at positions along the genome, given the available marker data. This can help us to identify regions where further markers might be added. In addition, as we will see in the next chapter, standard interval mapping to identify regions harboring QTL can occasionally give spurious evidence for linkage in regions of low genotype information, and so an evaluation of the proportion of missing information in regions of inferred QTL can help us to identify such problems.

Consider a fixed position in the genome and let g_i denote the genotype of individual i at that site. We first calculate $p_{ij} = \Pr(g_i = j \mid \mathbf{M}_i)$, where $\mathbf{M}_i$ denotes the multipoint marker genotype data for individual i . We consider two methods for defining the proportion of missing genotype information. First, we correspond the possible genotypes with integers (1 and 2 for a backcross, and 1, 2, and 3 for an intercross), and calculate the conditional variance of the genotypes, given the available marker genotype data, $\sum_i \text{var}(g_i \mid \mathbf{M}_i)$, and look at the ratio of this variance to the variance in the case of no genotype information ($n/4$ for a backcross and $n/2$ for an intercross). If there is complete genotype information (e.g., at a fully typed marker), we obtain a ratio of 0; if there is no genotype information, we obtain a ratio of 1.

In the second method, we use the information theoretic concept of entropy, $-\sum_i \sum_j p_{ij} \log_2 p_{ij}$, where we take $0 \log 0 = 0$. We again take the ratio of this quantity to the value for the case of no genotype information (n for a backcross and $3n/2$ for an intercross). Again, if there is complete genotype information, we obtain a ratio of 0; if there is no genotype information, we obtain a ratio of 1.

These quantities may be calculated and plotted with `plot.info`. For example, a plot of the missing genotype information in the `hyper` data appears in Fig. 3.14, which was created with the following.

```
> plot.info(hyper, col=c("blue", "red"))
```

We use `col` to indicate that the entropy and variance versions of the results should be plotted in blue and red, respectively.

The proportion of missing genotype information is effectively 0 at the fully typed markers. For several chromosomes, the minimal missing information is

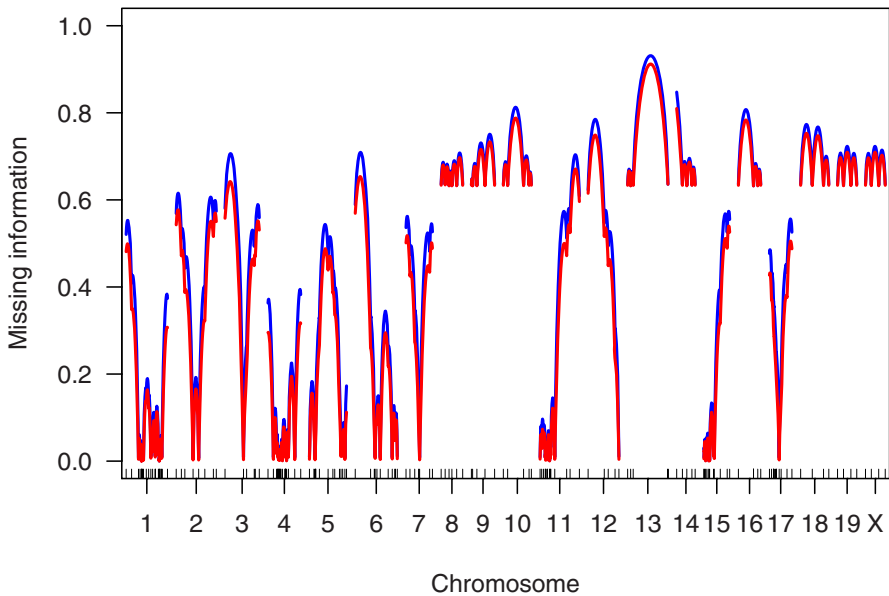


Figure 3.14. The proportion of missing genotype information in the `hyper` data. Results by the entropy and variance versions are shown in blue and red, respectively.

about 63%, as only the 92 individuals (out of 250) with extreme phenotypes were genotyped.

The detailed results of `plot.info` may be saved in an object. We can get the results just at the markers by rerunning `plot.info` with `step=0` and then show the results just for chromosome 14 as follows. (The argument `step` indicates the density of the grid, in cM, at which the missing information is to be calculated. The default value is `step=1`; use of `step=0` indicates that the calculation should be performed only at the markers.)

```
> z <- plot.info(hyper, step=0)
> z[ z[,1]==14, ]
```

	chr	pos	misinfo.entropy	misinfo.variance
D14Mit48	14	0.00	0.8475	0.8098
D14Mit14	14	16.40	0.6355	0.6335
D14Mit37	14	29.05	0.6331	0.6324
D14Mit7	14	43.68	0.6344	0.6333
D14Mit266	14	52.97	0.6355	0.6336

The result is a matrix with four columns: the chromosome and cM position followed by the missing information results by the entropy and variance methods, respectively.

A related function of interest is `nmissing`, which returns the number of missing genotypes for each individual or each marker (according to whether

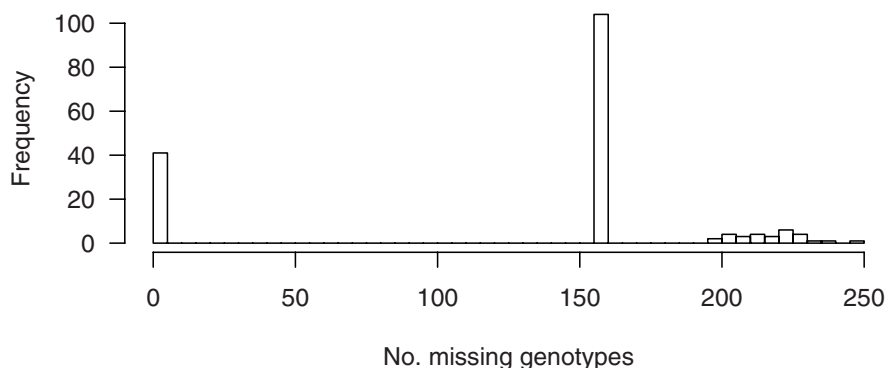


Figure 3.15. Histogram of the number of individuals with missing genotypes at the markers in the `hyper` data.

the argument `what` is "ind" or "mar", respectively). For example, we can get a histogram of the number of missing genotypes at the markers in the `hyper` data as follows. The results are in Fig. 3.15.

```
> hist(nmissing(hyper, what="mar"), breaks=50)
```

About 40 markers were typed on essentially everyone; over 100 were typed on only the 92 individuals with extreme phenotypes. The remaining 29 markers were typed only on a few individuals.

There is also a function `ntyped` which provides the opposite of `nmissing`: the number of typed markers for each individual, or the number of typed individuals for each marker.

3.8 Summary

Success in QTL mapping requires high-quality data. Given the time and expense in gathering data, reasonable effort should be devoted to identifying and correcting errors in the data prior to QTL analysis.

Histograms, scatterplots and time-course plots can assist in the identification of gross errors in the phenotype data, or of real but odd individuals deserving careful consideration in later analysis.

The assessment of genotype data begins with the inspection of the segregation patterns. Problem markers are often revealed by departures from the 1:1 and 1:2:1 patterns expected in a backcross and intercross, respectively.

While the availability of sequence-based marker maps in many organisms has eliminated much of the effort that was once required to establish marker order, the genetic maps of typed markers should still be carefully inspected. Errors in marker labels are not uncommon, and these may be revealed by an inspection of pairwise linkages and the reestimation of intermarker genetic distances.

Finally, it can be useful to study the pattern of crossovers to identify genotyping errors that are revealed by apparent tight double crossovers. The calculation of genotyping error LOD scores simplifies this effort. However, the presence of a small number of genotyping errors will not have much influence on later results, and so this aspect of the detective work is generally not critical.

3.9 Further reading

Surprisingly little has been written on the detective work involved in identifying and resolving errors in QTL mapping data. Of some relevance is Broman (1999), concerning cleaning human genotype and pedigree data. The genotyping error LOD scores were developed by Lincoln and Lander (1992).