QTL Mapping Tutorial

John T. Lovell

2017-10-23

Contents

---

email: johntlovell@gmail.com – website: lovelleeb.weebly.com – github: github.com/jtlovell/qtlTools

---

1 Setup

For this tutorial, we need a couple datasets and R packages.

1. The `R/qtl` R pacakge.

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

2. The `qtlTools` and `qtlToolsTutorials` R packages, in development.

- If you don't have the devtools package, run:
    - `install.packages("devtools")`
- `devtools::install_github("jtlovell/qtlTools")`
- `devtools::install_github("jtlovell/qtlToolsTutorials")`
- `library(qtlTools)`
- `library(qtlToolsTutorials)`

3. The `ggplot2` R package

- `install.pacakges("ggplot2")`
- `library(ggplot2)`

The Tutorial dataset is included with the qtlToolsTutorials package

```
data("completeQTL_tutorial_data")
```

2 Overview: Concept of QTL mapping

In essence, QTL mapping is the correlation of phenotype data with genotype data. For each genetic marker, a statistical test is applied. In the simplest form, this is just a linear model. However, as mentioned in the genetic map construction tutorial, genetic data is error prone and often incomplete. Therefore, QTL mapping usually falls into two steps: 1) Figure out the genotype probabilites along an evenly spaced "pseudomarker" grid. This is often accomplished using a HMM. 2) Apply a likelihood-ratio statistic at each marker or pseudomarker, testing the odds ratio of a QTL present (Ha) vs. no QTL (Ho). The log of this odds ratio is known as the LOD score and is the most commonly reported QTL test statistic.

Here, we will follow a relatively standard QTL mapping work flow, as implemented in R/qtl… 1. Calculate genotype probabilities (`qtl::calc.genoprob`), filling in gaps in the map. 2. Run one-way QTL scans 3. Test significance of QTL peaks using permutations 4. Look for QTL*covariate interactions 5. Look for additional QTL

3 Formatting and loading the data

We will analyze a single trait in an F2 P. hallii mapping population. For the purposes of this tutorial, the data is stored in a standard `R/qtl cross` object: `cr`.

```
summary(cr)
```

```
##     F2 intercross
##
##     No. individuals:    244
##
##     No. phenotypes:     3
##     Percent phenotyped: 100 100 100
##
##     No. chromosomes:    9
##         Autosomes:      1 2 3 4 5 6 7 8 9
##
##     Total markers:      884
##     No. markers:        105 127 121 75 119 95 65 49 128
##     Percent genotyped:  99.8
##     Genotypes (%):      AA:22.2  AB:50.5  BB:27.3  not BB:0.0  not AA:0.0
```

However, typical QTL analysis datasets start with a genotype matrix, a `map`, and a phenotype matrix. To illustrate how to get the data into the right format, lets extract the map and the genotype / phenotype matrices from `cr`.

```
phe.mat<-pull.pheno(cr)
geno.mat<-pull.geno(cr)
map<-pullMap(cr)
```

To simplify things, we store the genetic map positions within the column (marker) names of the genotype matrix and the line IDs as the row names.

```
colnames(geno.mat)<-paste0(map$chr,"_", round(map$pos,4))
rownames(geno.mat)<-getid(cr)
```

Table 1: The first 5 markers and F2 lines in the genotype matrix. Genotypes are coded as 1 = A/A, 2 = A/B, 3 = B/B.

	1_0	1_0.6089	1_1.2178	1_1.8267	1_2.6393
1	3	3	3	3	3
2	1	1	1	1	1
3	2	2	2	2	2
4	1	1	1	1	1
5	1	1	1	1	1

We can use a qtlTools function to write this matrix to an R/qtl input file, then use R/qtl::read.cross to load the file as a cross object. Note that we specify the coding of the genotypes and the type of cross when we read the matrix back in. This means that our genotype specification can be flexible.

```r
geno2cross(geno.mat, crossfile = "~/Downloads/cross4tut.csv")
```

```
## cross file written to ~/Downloads/cross4tut.csv
```

```r
cross<-read.cross("csv", file="~/Downloads/cross4tut.csv",
                  genotypes=c(1,2,3), crosstype="f2")
```

```
##  --Read the following data:
##   244  individuals
##   884  markers
##   1  phenotypes
##  --Cross type: f2
```

4 Pseudomarker calculation / imputatation

For a lucky few, the genotype matrix will be completely saturated (all recombination events are captured) and error-free. However, for the rest of us, we need to fill in gaps in the map and fix genotyping errors before conducting QTL analysis. The way that we undertake this task depends on the amount of missing data and the likelihood of genotyping errors.

A good way to get a sense of the amount of missing data is to look at the 'entropy' in your map. Higher values mean more missing data or larger gaps.

```r
plot.info(cross)
```

A general rule of thumb is that if entropy is high, impute missing genotypes using `sim.geno`. However, if it is low, calculate genotype probabilities using `calc.genoprob`. The benefit of imputations is that it does a far better job of accurately measuring QTL positions within large gaps. However, it can be an order of magnitude slower than using genotype probabilites. Both functions fill gaps in the map with 'psuedomarkers' that contain either the conditional genotype probabilities for that position in the map, or a matrix of `n.draws` imputations. Psuedomarkers fall along a grid where the distance between positions = `step`. Also, to speed things up, we can tell R/qtl to only fill gaps (and not generate an even grid) by specifying the `stepwidth` argument.

To get imputed pseudomarkers:

```r
cross<-sim.geno(cross, step = 1, stepwidth = "max", n.draws = 64,
                map.function = "kosambi", # use for plants
                error.prob = 0.0001) # set this for your genotyping platform.
```

To get conditional genotype probabilities:

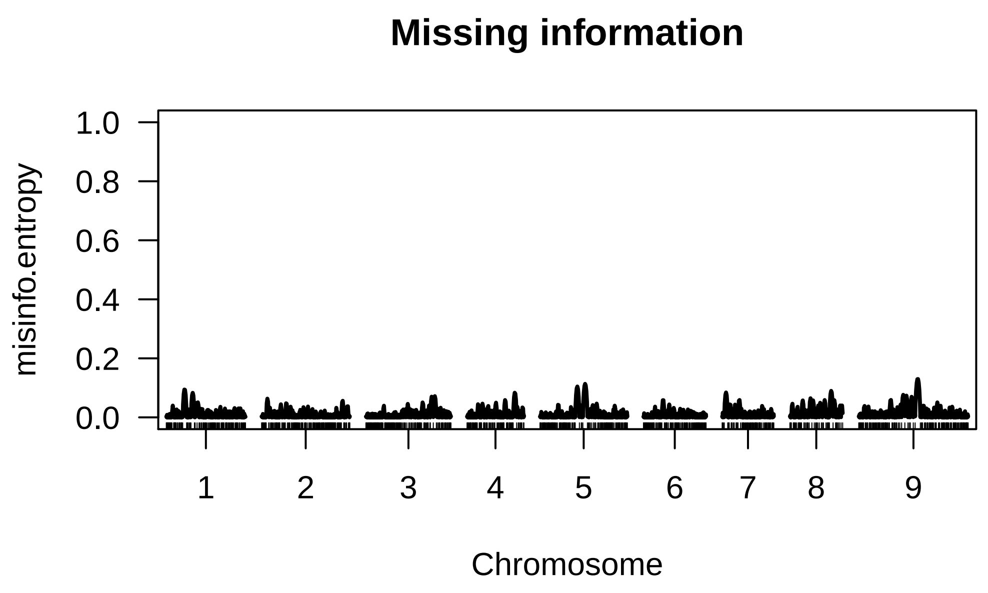


Figure 1: The scale of missing data in the cross. Note that the level of entropy in our cross does not exceed 20%. Not too shaby.

```
cross<-calc.genoprob(cross, step = 1, stepwidth = "max",
                map.function = "kosambi", # use for plants
                error.prob = 0.0001) # set this for your genotyping platform.
```

5 Basic QTL mapping.

To run a single-trait QTL scan, we use the R/qtl function `scanone`, which tests the likelihood of a QTL vs. the NULL that there is no QTL.

But first, we need to add phenotype data to the cross object. Lets make sure that the id's in the phenotype matrix match exactly the cross object ids:

```
identical(phe.mat$id, as.numeric(getid(cross)))
```

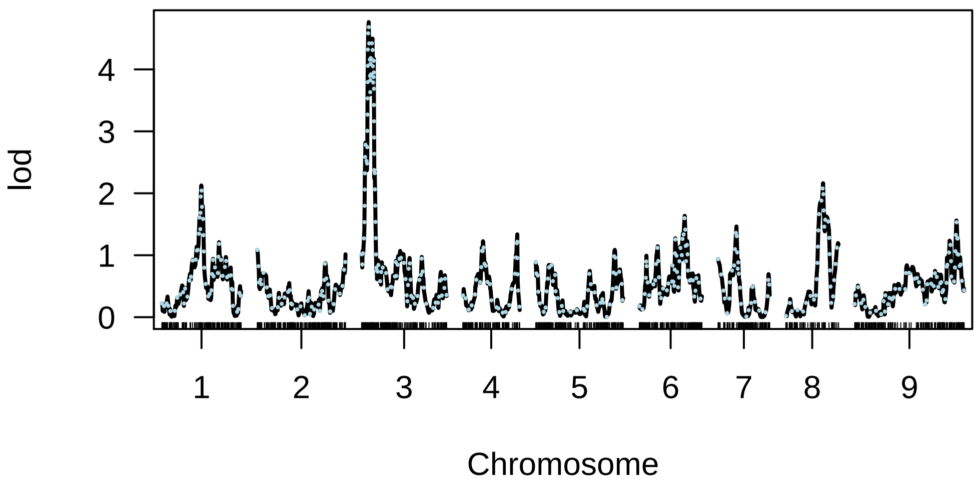
```
## [1] TRUE
```

Since they match, we can just add the phenotype matrix to the cross object as such:

```
cross$pheno<-phe.mat
```

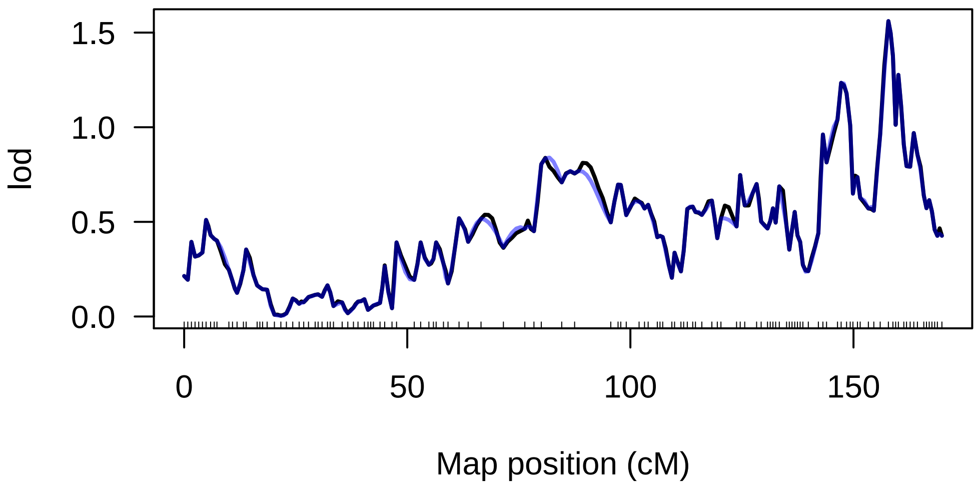
If we want to use imputations, we specify method = "imp", otherwise, I typically use method = "hk", which runs a regression on the conditional genotype probabilites at each (pseudo)marker. Note that in this case, since we have a dense marker grid, the method doesn't matter much.

```
s1.imp<-scanone(cross, method = "imp", pheno.col = "phenotype")
s1.hk<-scanone(cross, method = "hk", pheno.col = "phenotype")
plot(s1.imp, s1.hk, lty = c(1,3), col = c("black", "lightblue"))
```



However, we do have a few gaps in the map on Chr09. Note that the methods are a bit different there:

```
plot(s1.imp, s1.hk, lty = c(1,1), col = c("black", rgb(0,0,1,.5)), chr = 9)
```



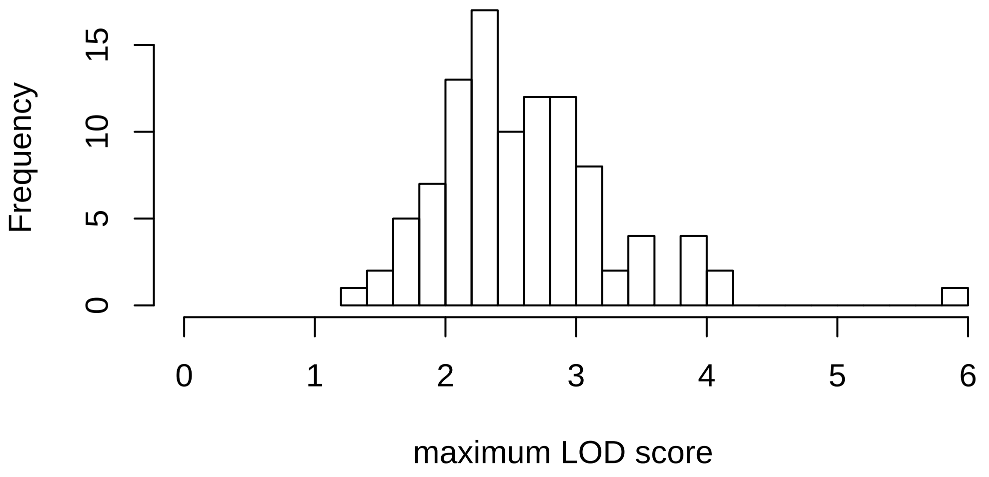
6 Permutation tests for significance

In GWAS and other QTL analyses, P-value adjustments (e.g. Bonferonni, qvalue, etc.) are employed to determine significance. However, in linkage mapping, we know that adjacent markers are not independent, thereby violating the assumptions of P-value transformations. As such, we run permutations to test for significance, which randomize the phenotype data, relative to the genotypes and output the maximum NULL LOD score. We then test if our QTL peak is higher than 95% (or whatever) of these permuted LOD scores

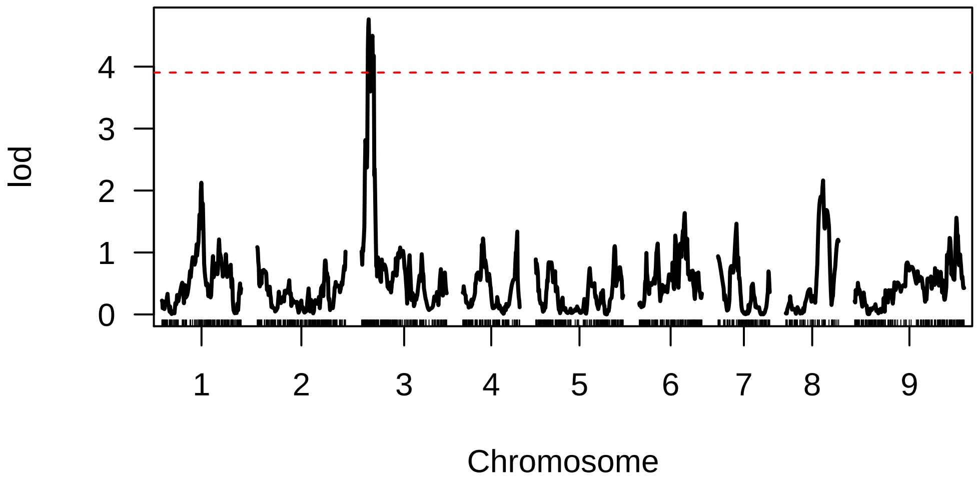
```
s1.perm<-scanone(cross, method = "hk", pheno.col = "phenotype", n.perm = 100)
```

```
## Doing permutation in batch mode ...
```

```
plot(s1.perm)
```



```
plot(s1.hk)
add.threshold(s1.hk, perms = s1.perm, col = "red", lty = 2)
```



```
summary(s1.hk, perms = s1.perm, alpha = 0.05, pvalues = T)
```

```
##           chr pos  lod pval
## 3_10.9819   3  11 4.76 0.01
```

7 QTL-covariate interactions.

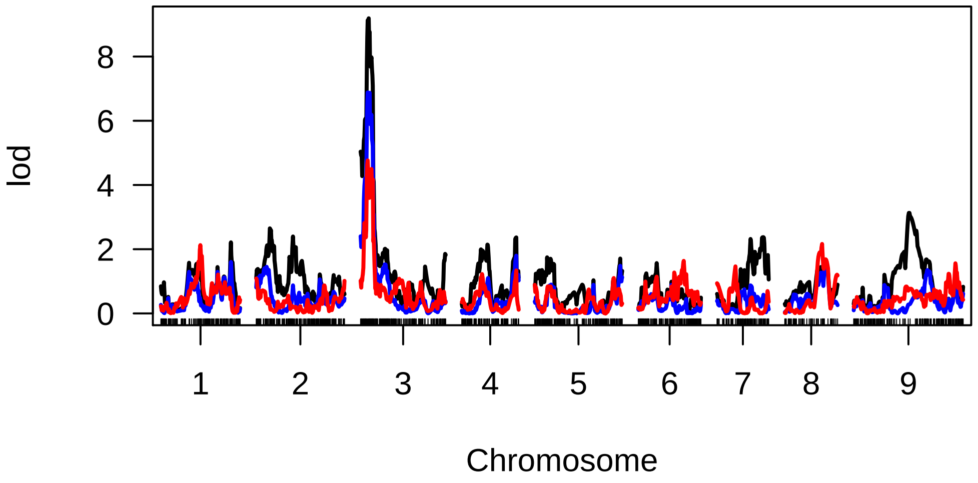
Often our experimental designs are complicated and require correction for experimental covariates. We may also be interested in how allelic effects at a QTL are modulated by a covariate. In this example, we have two treatments `wet` and `dry`. To use this information, we need to make a data.frame that contains numeric-coded covariate identities for each individual:

```
covar = data.frame(covar = as.numeric(as.factor(pull.pheno(cross, "Treatment"))))
head(covar)
```

```
##   covar
## 1     1
## 2     2
## 3      1
## 4      2
## 5      2
## 6      2
```

We can then include this covariate in the QTL scan and compare the effects of the covariate by running three scans: no covariate, additive covariate and an interactive covariate

```
s1.no.covar<-scanone(cross, method = "hk", pheno.col = "phenotype")
s1.add.covar<-scanone(cross, method = "hk", pheno.col = "phenotype", addcovar = covar)
s1.int.covar<-scanone(cross, method = "hk", pheno.col = "phenotype", addcovar = covar, intcovar = covar
plot(s1.int.covar, s1.add.covar,s1.no.covar)
```



We can now use permutations to test if the interaction or additive effect are significant:

```
set.seed(42)
perm.no.covar<-scanone(cross, method = "hk", pheno.col = "phenotype",
                       n.perm = 100, perm.strata = covar[,1], addcovar = NULL, verbose = F)
set.seed(42)
perm.add.covar<-scanone(cross, method = "hk", pheno.col = "phenotype",
                       n.perm = 100, perm.strata = covar[,1], addcovar = covar, verbose = F)
set.seed(42)
perm.int.covar<-scanone(cross, method = "hk", pheno.col = "phenotype",
                       n.perm = 100, perm.strata = covar[,1],
                       addcovar = covar, intcovar = covar, verbose = F)
kable(summary(s1.add.covar-s1.no.covar,
        perms = perm.add.covar-perm.no.covar,
        pvalues = T), caption = "scanone results for the difference between no covariate andadditive co
```

Table 2: scanone results for the difference between no covariate andadditive covariate. Note that the Chr03 QTL is significant.

	chr	pos	lod	pval
1_108.6226	1	108.62260	1.0220600	0.16
c2.loc11	2	16.88775	1.0066822	0.16
3_13.4211	3	13.42110	2.6598136	0.01
4_87.6469	4	87.64690	0.9908900	0.16
5_132.6055	5	132.60550	0.8649872	0.19
6_5.0832	6	5.08320	0.2561507	0.39
7_42.4132	7	42.41320	0.7477194	0.21

	chr	pos	lod	pval
8_13.7904	8	13.79040	0.5229474	0.29
9_111.939	9	111.93900	0.8485483	0.20

```r
kable(summary(s1.int.covar-s1.add.covar,
        perms = perm.int.covar-perm.add.covar,
        pvalues = T), caption = "scanone results for the difference between interactive covariate and ad
```

Table 3: scanone results for the difference between interactive covariate and additive covariate. Note that the Chr03 QTL marginally significant.

	chr	pos	lod	pval
1_4.6697	1	4.6697	0.9524186	0.85
2_22.9075	2	22.9075	1.9996064	0.26
3_1.0175	3	1.0175	2.8445323	0.12
4_29.7485	4	29.7485	1.2496078	0.75
5_11.2018	5	11.2018	1.3170476	0.66
6_28.5262	6	28.5262	0.9327814	0.85
7_69.7423	7	69.7423	1.9393304	0.29
8_22.4096	8	22.4096	0.7062760	0.91
c9.loc57	9	85.5772	2.8539506	0.11

```r
effectplot(cross, pheno.col = "phenotype", mname1 = "3_1.0175",
           mark2 = covar[,1], geno2 = c("dry","wet",""))
```

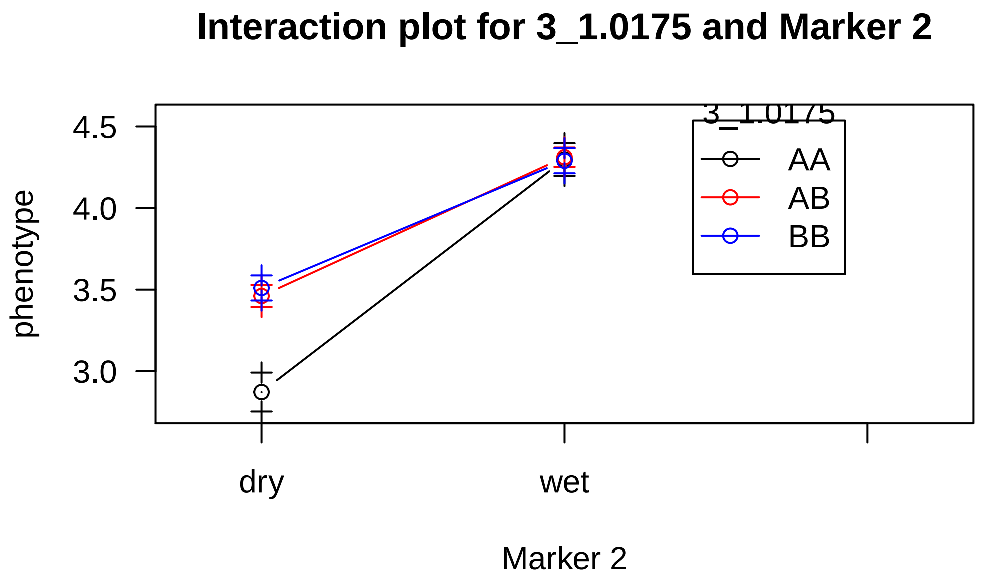
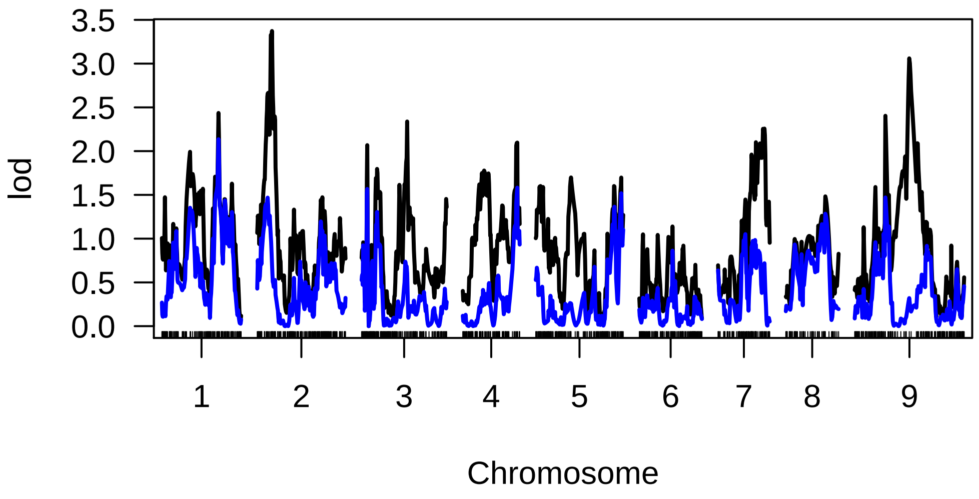


Figure 2: interaction between treatment and genotype. Note that the genotype effect only slightly depends on the treatment.

8 Find additional QTL

Like in general statistical modeling, if we can control for variation in the response variable, we have greater ability to detect effects of predictors (markers) in QTL mapping. As such, it can be powerful to build multiple QTL models.

```r
qtl = makeqtl(cross, chr = 3, pos = 11, what = "prob")
scan4secondqtl_add<-addqtl(cross, pheno.col = "phenotype", covar = covar,
                           formula = "y~Q1+Q1*covar+Q2+covar", qtl = qtl,
                           method = "hk")
scan4secondqtl_int<-addqtl(cross, pheno.col = "phenotype", covar = covar,
                           formula = "y~Q1+Q1*covar+Q2*covar+covar", qtl = qtl,
                           method = "hk")
plot(scan4secondqtl_int, scan4secondqtl_add)
```



In this case, there is not a lot going on - just a single QTL, but this is not always the case.