Tutorial - Multiple QTL Model (MQM) Analysis

Danny Arends, Pjotr Prins, Karl Broman and Ritsert Jansen

January 12, 2010

1 Introduction

Multiple QTL model (MQM) mapping is a powerful QTL mapping method developed by Ritsert C. Jansen[8]. MQM is an automatic two-stage procedure in which, in the first stage, important markers are selected in multiple regression on markers. In the second stage a QTL is moved along the chromosomes by using the pre-selected markers as cofactors, except for the markers flanking the interval under study. A refined and automated procedure for cases with large numbers of marker cofactors is included.

MQM for R/qtl is based on the original implementation of MQM mapping and consists of a three step procedure: (1) data augmentation, (2) automatic backward model elimination using genetic markers evenly spread over the genome as cofactors and (3) QTL (interval) mapping using the most 'informative' model through maximum likelihood. The method internally controls false discovery rates (FDR) and lets users test different QTL models by elimation of non-significant cofactors.

R/qtl-MQM has the following advantages

- Higher power, as long as the QTL explain a reasonable amount of variation
- Protection against overfitting, because it uses the residual variance from the full model. For this reason more parameters (cofactors) can be used compared to, for example, CIM
- Prevention of ghost QTL (between two QTL in coupling phase)
- Detection of negating QTL (QTL in repulsion phase)

The current implementation of R/qtl-MQM has the following recognized limitations: (1) MQM is limited to experimental crosses F2, RIL and BC. (2) MQM augmentation drops entire individuals from the equation when a lot of marker data is missing. (3) MQM does not treat sex chromosomes differently from autosomal chromosomes - though one can introduce sex as a cofactor. Finally, (4) automatic backward elimination sets cofactors at fixed locations and ignores underlying known marker density. Future versions of R/qtl-MQM may improve on these points. Check the changelog for updates.

Despite these limitations, *MQM*[1] is a valuable addition to the QTL mapper's toolbox. It is able to deal with interference of QTL, **[I'm not sure you mean by "interference of qtl"]** handles missing data and allows more precise detection than other methods. This tutorial will show how to use *MQM* for QTL mapping. Also R/qtl's *MQM* is faster than other implementations and scales on multi-CPU systems and computer clusters.

MQM is an integral part of the free *R/qtl* package[2, 1, 3] for the R statistical language[2].

2 A quick overview of *MQM*

These are the typical steps in an *MQM* QTL analysis

- Load an experiment using R/qtl data types

[1] Note that *MQM* [8] is *not the same* as composite interval mapping (CIM)[13, 14], though both methods were introduced around the same time. The great advantage of MQM compared to interval mapping in reducing the chance of a type I error (a QTL is indicated at a location where there is no QTL present) and in reducing the chance of a type II error (a QTL is not detected)[9]

[2] We assume the reader knows how to load his data into R using the R/qtl `read.cross` function, see also the R/qtl tutorials[1] and book[2]

- Fill in missing data, using either `fill.geno` or `mqmaugmentdata`
- Unsupervised backward elimination to analyse *cofactors*, using `mqmscan`
- Optionally select *cofactors* at markers that are thought to influence QTL at, or near, the location
- Permutation or simulation analysis to get estimates of significance, using `mqmpermute`

Using maximum likelihood (ML), or restricted maximum likelihood (REML), the algorithm employs a backward elimination strategy to identify QTL underlying the trait. The algorithm passes through the following stages:

- Calculation of relative marker positions and detection of linkage groups
- Optional (Re-)estimation of genetic map and/or recombination frequencies
- Likelihood estimations of the full model using all cofactors
- Backward elimination of cofactors, followed by a genome scan for QTL
- If there are no *cofactors* defined, perform a genome scan testing each genetic location independently

The interval map created during the genome scan and the QTL model are returned as an (*MQM* extended) R/qtl `scanone` object. Several special plotting routines are available for *MQM* results.

3 Data augmentation

Most real world datasets are incomplete. That is, genotype information is missing, or can have multiple plausible values. *MQM* automatically expands the dataset by adding all potential variants and attaches a probability. For example, information is missing (unknown) at a marker location for one individual. Based on the values of the neighbouring markers, and the (estimated) recombination rate, a probability is attached. With *MQM* all likely options are considered. When genotypes A and B are possible at the location two 'individuals' are created in the augmentation step, one with genotype A, and one with genotype B. A probability is attached to either variant. The combined probabilities of all markers tells us whether a combination of variants is likely, or not.

To see an example of missing data with an F2 intercross, we can visualize the genotypes of the individuals using `plot.geno` In figure 1 there are 2% missing values in white. The other colors are genotypes at a certain position, for a certain individual. Simulate an F2 dataset with 2% missing markers as follows:

Simulate a dataset with missing data:

```
> library(qtl)
> data(map10)
> mycross <- sim.cross(map10, type = "f2", n.ind = 100, missing.prob = 0.02)
```

```
> geno.image(mycross)
```

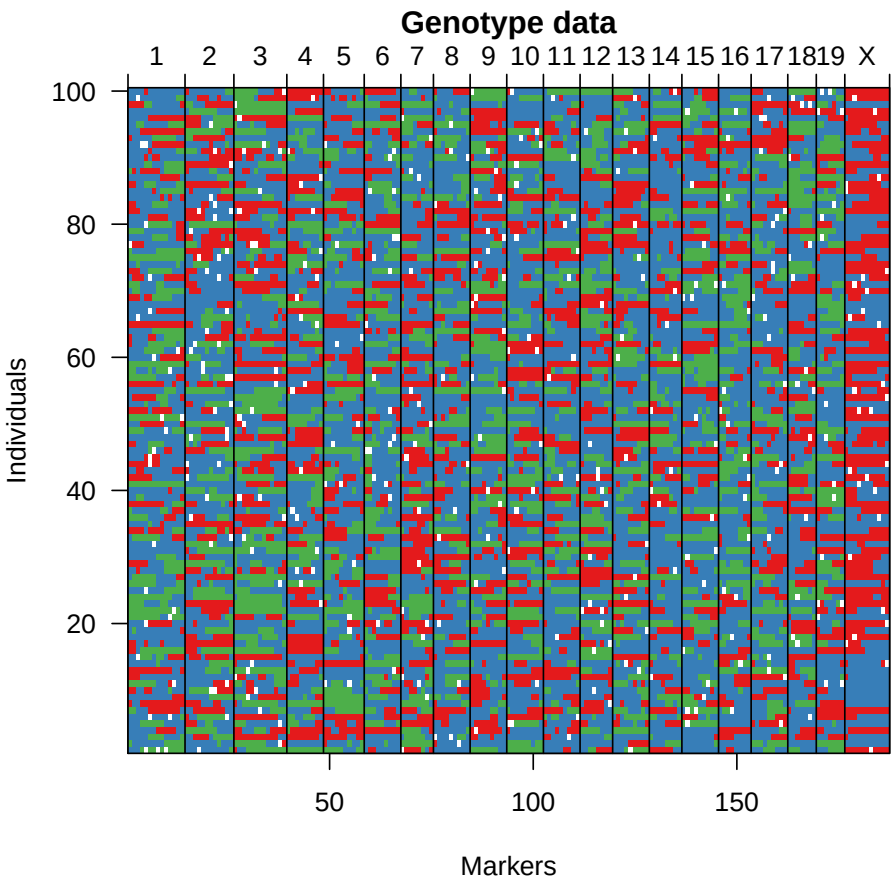

Figure 1: Simulate an F2 cross with 2% missing data starting from the map10 F2 intercross. The figure displays genotypes of 100 individuals with around 2% missing data (white) for the map10 F2 intercross

Before we can go to the next step (QTL genome scan), the data has to be completed (i.e. no more missing data). There are two possibilities: either use the imputation function `fill.geno` or use the *MQM* data augmentation routine `mqmaugment`. With `fill.geno` a single 'most likely' genotype is filled in using multiple imputations, estimating the missing marker genotypes. In contrast, *augmentation* introduces multiple likely genotypes. At this stage `mqmaugment` is specific to *MQM* and the recommended procedure[3].

In this tutorial we use *MQM*'s augmentation. The function `mqmaugment` fills in missing genotypes for us. For each missing genotype data, at a marker, it fills in all possible genotypes and calculates the probability. When the total probability is more likely than the `minprob` parameter the *augmented* individual is stored in the new cross object, ready for QTL mapping.

The important parameters are: `cross`, `pheno.col`, `maxaugind`, `minprob` and `verbose` (see also the `mqmaugment` help page in R). `maxaugind` sets the maximum number of *augmented* genotypes per individual in a dataset. The default of 82 allows six missing markers per individual

[3]Note that after augmentation the resulting cross object is no longer suitable for the use with `scanone` or *CIM*, because of the additional information stored

in a BC, and four in an F2. As a result the user has to increase the `maxaugind` parameter when there are more missing markers.

The `minprob` parameter sets the minimum probability of a genotype for inclusion in the augmented dataset. This genotype probability is calculated for every marker *relative* to the most likely individual. Note that setting this value too low may result in dropping individuals entirely as `maxaugind` is quickly reached footnoteThe current version drops individuals that go beyond `maxaugind`. This is undesirable behaviour and will be fixed in an upcoming release. A value `minprob=1.0` may prevent the dropping of individuals, making augmentation behave similar to `fill.geno`'s imputation method. Use the verbose option to get more feedback on the augmentation routine and check how many individuals are dropped. So first we try augmentation with `minprob=1.0` (figure 2):

Plot augmented data using `plot.geno` *with* `minprob=1.0`*:*

```
> augmentedcross <- mqmaugment(mycross, minprob = 1)

Starting C-part of the data augmentation routine
Filling the chromosome matrix
```

With a lower `minprob`, more *likely* genotypes considered, and the resulting *augmented* dataset will be larger. The (weighted) augmented individuals with all possible genotypes theoretically leads to more accurate mapping when dealing with missing values [11][4]. Try augmentation with `minprob=0.1` (figure 3):

[4]Note that the augmented dataset can only be used with *MQM* functions. *MQM* functions recognise expanded individuals as single entities. Other R/qtl functions, like `scanone`, assume the augmented individuals are *real* individuals

```
> plot.geno(augmentedcross)
```

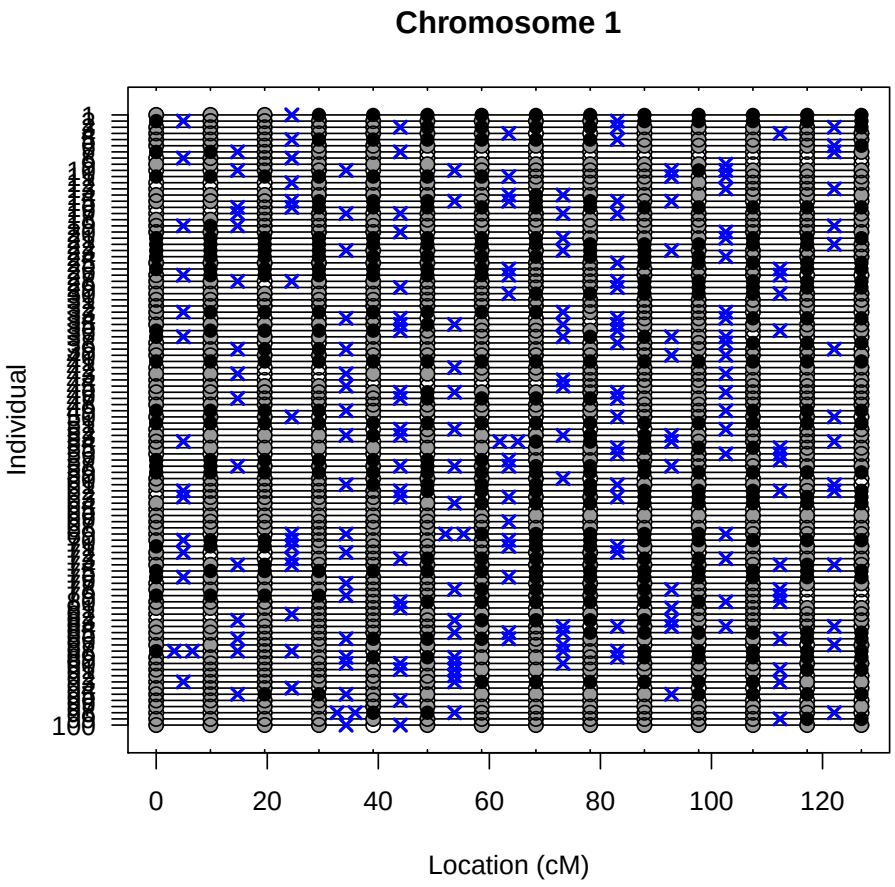

Figure 2: `plot.geno` displays the genotypes of 100 filled individuals (`mqmaugment` with `minprob=1` means only the 'most likely' individual is used and no real expansion of the dataset takes place, similar to `fill.geno`)

```
> augmentedcross <- mqmaugment(mycross, minprob = 0.1)

Starting C-part of the data augmentation routine
Filling the chromosome matrix

> plot.geno(augmentedcross)
```

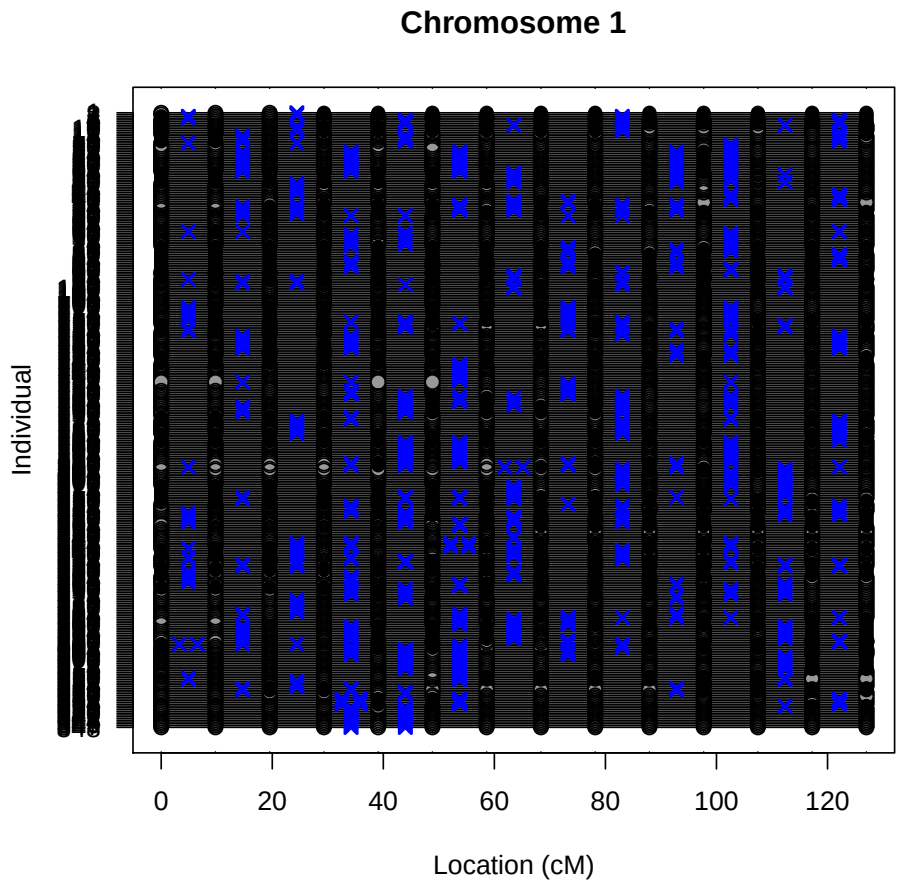

Figure 3: `plot.geno` displays the *augmented* genotypes of 100 individuals. There are a total of 340 'expanded' individuals in this plot, because *MQM* fills in missing markers with all likely genotypes (an average expansion of 3.4 per individual)

4 MQM multiple QTL model mapping

The famous mouse hyper tension dataset, which comes standard with R/qtl, is a backcrossed (BC) offspring of mice with two listed traits: bloodpressure (bp) and sex. Here we show an example of running both `scanone` single marker QTL analysis and `mqmscan` multiple QTL model mapping, to map QTL for blood pressure. First, *complete* missing data:

Use `scanone` and `mqmscan` after filling missing data with `mqmaugment`:

```
> data(hyper)
> colors <- c("Black", "Green")
> lines <- c(2, 1)
> hyperaug <- mqmaugment(hyper, minprob = 1)

Starting C-part of the data augmentation routine
Filling the chromosome matrix

> result_scanone <- scanone(hyperaug)
> result_no_cofactors <- mqmscan(hyperaug)
```

After loading `data(hyper)` we create a dataset without missing genotypes using the `mqmaugment` routine discussed in section 3. Next we scan for QTL using both `scanone` and `mqm` routines (using default parameters). Figure 4 shows that with `minprob=1.0` there is virtually no difference in outcome.

MQM introduces pseudo markers by default (unlike `scanone`). A pseudo marker has a name like c7.loc25 - i.e. the 25 cM pseudo marker on chromosome 7. This reflects the internally used interval mapping. The chromosomes are divided into equally spaced sections. Therefore each chromosome is divided into fictional markers spaced equally `stepsize` cM apart. A LOD score for underlying QTLs is calculated at these fictional pseudo markers. A small `stepsize` allows for smoother profiles compared with pure marker based mapping approaches. The real markers are listed between the pseudo markers. In the result you can remove the pseudo markers by using the function `mqmextractmarkers(res)`.

For (automatic) model selection in *MQM* we first need to supply the algorithm with an initial model. This initial model can be produced in two ways: by (1) building a model by hand (forward stepwise), or (2) by unsupervised backward elimination on a large number of markers.

First we will demonstrate building the initial model by hand using a forward stepwise approach (note that the automated procedure is preferred, both for theoretical and practical reasons). A model consists of a set of markers we want to account for. We can start building the initial model by adding cofactors at markers with high LOD scores. figure 4 displayed a large QTL peek on chromosome 4 at 30 cM, so let us account for that by setting a cofactor at the marker nearest to the peek on chromosome 4 and perform a new *MQM* scan (result in figures 5 and 6):

Add marker D4Mit164 as a cofactor:

```
> summary(result_no_cofactors)

          chr pos (Cm) LOD bp  info LOD*info
c1.loc80    1       80  3.534 0.888    3.139
c2.loc60    2       60  1.448 0.907    1.313
c3.loc25    3       25  0.612 0.913    0.559
```

```
> plot(result_no_cofactors, result_scanone, col = colors, lwd = lines)
```

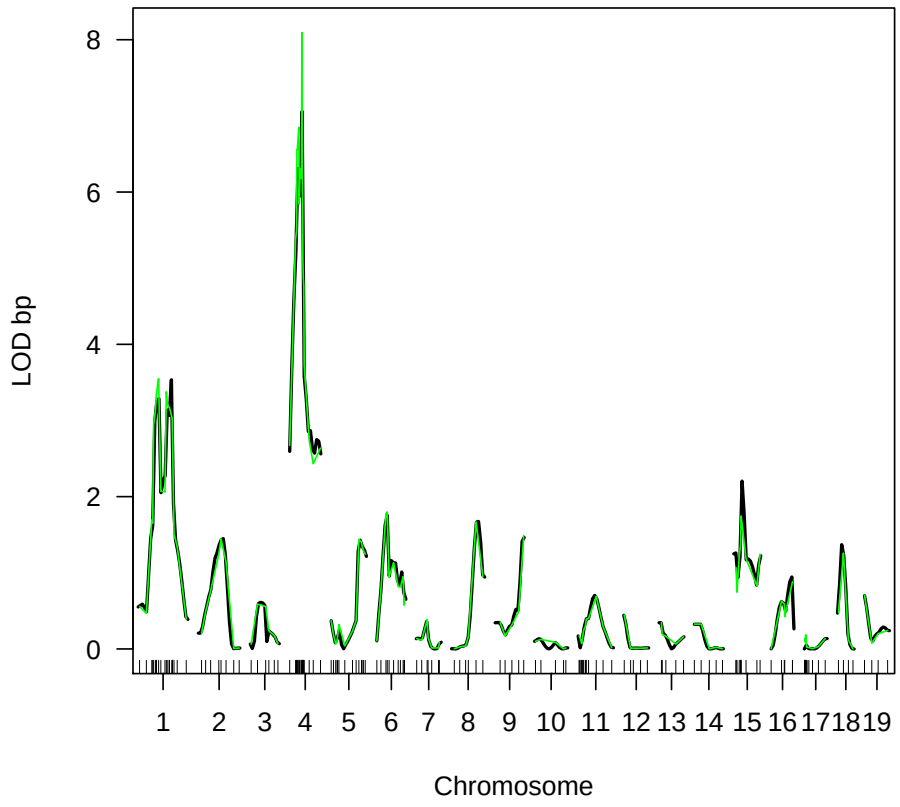

Figure 4: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice comparing *MQM* (black) and single QTL mapping with `scanone` (green). Both routines are virtually equal when used without *augmentation* and no extra parameters

```
c4.loc30    4       30  7.059 0.928    6.550
c5.loc70    5       70  1.428 0.937    1.338
c6.loc25    6       25  1.755 0.953    1.673
c7.loc25    7       25  0.361 0.944    0.341
c8.loc65    8       65  1.674 0.908    1.521
c9.loc70    9       70  1.464 0.933    1.366
c10.loc10  10       10  0.134 0.912    0.122
c11.loc40  11       40  0.703 0.909    0.639
c12.loc0   12        0  0.442 0.933    0.412
c13.loc0   13        0  0.344 0.849    0.292
c14.loc10  14       10  0.324 0.942    0.305
c15.loc20  15       20  2.204 0.929    2.047
c16.loc50  16       50  0.945 0.942    0.890
c17.loc55  17       55  0.136 0.867    0.118
c18.loc10  18       10  1.369 0.892    1.222
D19Mit59   19        0  0.700 0.954    0.668
```

```
> find.marker(hyperaug, chr = 4, pos = 30)

[1] "D4Mit164"

> toset <- which.marker(hyperaug, "D4Mit164")

Marker D4Mit164 is number 46

> cofactorlist <- mqmcofactors(hyperaug, toset)
> result_1 <- mqmscan(hyperaug, cofactorlist)
```

Plot after adding cofactor:

Figures 5 and 6 show the effect of setting a single marker as a cofactor related to the QTL on chromosome 4, followed by an *MQM* scan. The marker is not dropped and it passes initial thresholding to account for the `cofactor.significance` level. LOD scores are expected to change (slightly) genome wide because of variation already explained by the QTL on chromosome 4. This can be shown by plotting these new results against the *MQM* QTL mapping without cofactors (figure 6).

Figure 6 clearly shows the second peek on chromosome 1 at 70 cM increases, so we add that to the model and check if the model with both cofactors changes the QTL. Again, combining `which.marker` with `find.marker`, adds the new cofactornumber `c` with the cofactor already in `toset` (see figure 7):

Add another cofactor on chromosome 1 at 70 cM:

```
> toset <- c(toset, which.marker(hyperaug, find.marker(hyperaug, 1, 70)))

Marker D1Mit218 is number 12

> cofactorlist <- mqmcofactors(hyperaug, toset)
> result_2 <- mqmscan(hyperaug, cofactorlist)
```

Plot after adding second cofactor:

When using the functions `mqmcofactors` and `mqmsetcofactors`, the number of cofactors is compared against the number of individuals inside the cross object. If there is a danger of setting too many cofactors the two cofactor functions will warn the user by displaying an error about the user trying to set more cofactors than `samplesize` permits, and not return a cofactorlist.

MQM verifies the `cofactor.significance` level specified by the user. The marker on chromosome 1 is informative enough to be included into the model. Thus creating a new initial model consisting of cofactors on chromosome 4 and 1. This (forward) selection of cofactors can continue until there are no more informative markers.

Manually determining which marker to set a cofactor can be very time consuming in the case of many QTL underlying a trait. It is also prone to overfitting (because the user manipulates the data). Furthermore manual fitting is generally not feasable for a large number of traits. Fortunately *MQM* provides unsupervised backward elimination.

```
> op <- par(mfrow = c(2, 1))
> plot(mqmgetmodel(result_1))
> plot(result_1)
```

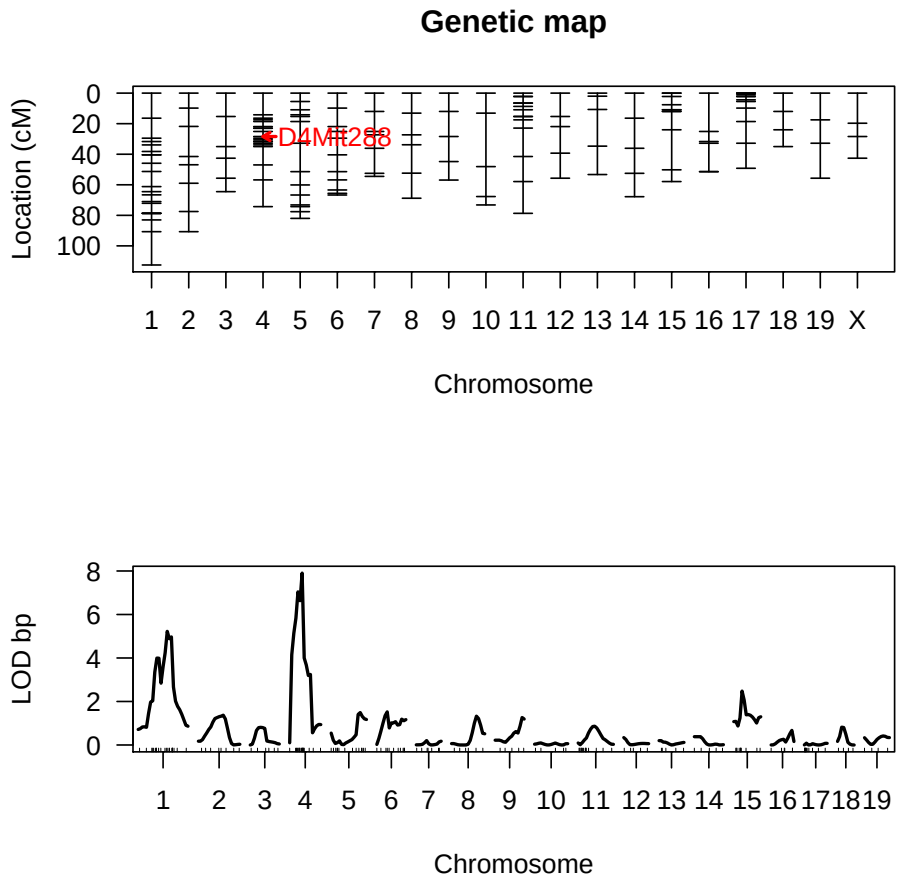

Figure 5: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice using *MQM*. A cofactor is added at chromosome 4 (D4Mit164) and kept in the model. The LOD score (evidence) for a second QTL on chromosome 1 increases (see also figure 6)

```
> plot(result_1, result_scanone, col = colors, lwd = lines)
```

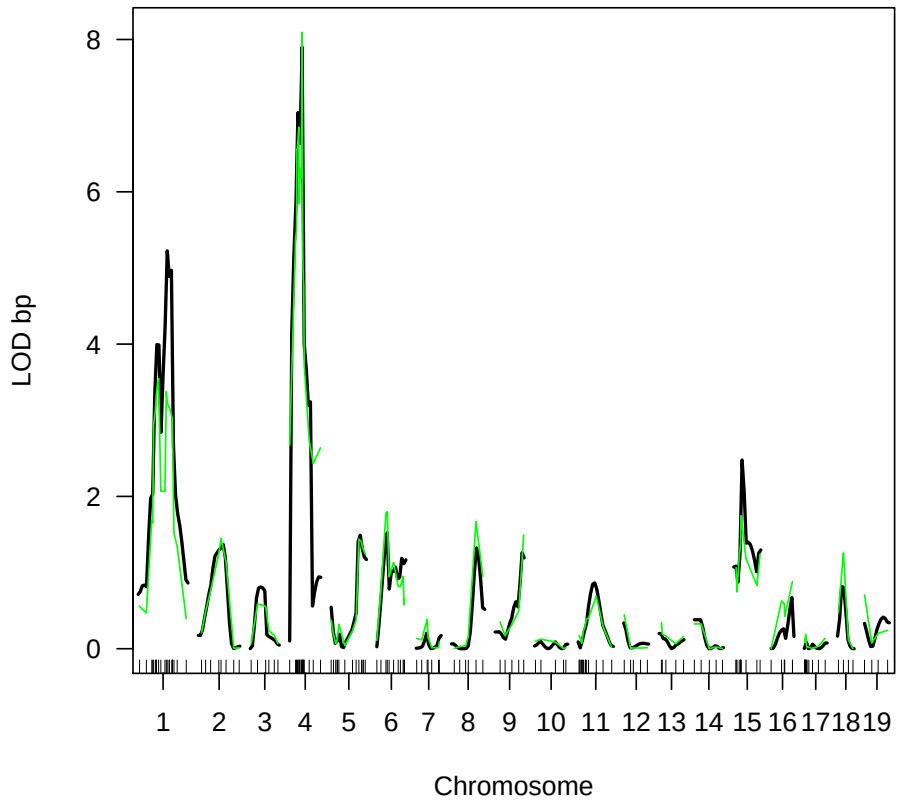

Figure 6: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice comparing *MQM* (black) and single QTL mapping with `scanone` (green), after introducing a cofactor at chromosome 4 (D4Mit164) accounting for variation explained at that location

```
> op <- par(mfrow = c(2, 1))
> plot(mqmgetmodel(result_2))
> plot(result_2, result_1, col = colors, lwd = lines)
```

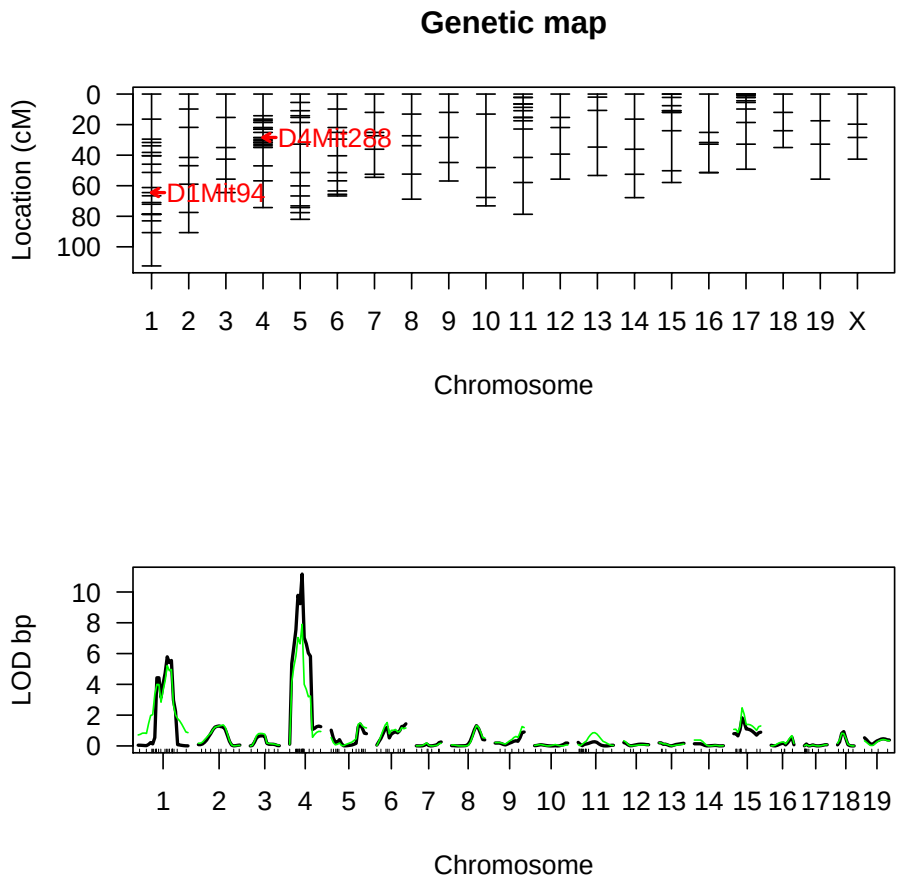

Figure 7: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice comparing *MQM* (black) and single QTL mapping with `scanone` (green), after introducing a cofactors at chromosomes 1 and 4 there appears only a small difference in the likelihood scores for each chromosome. The LOD scores do not appear to improve much after adding the second cofactor on chromosome 4. But... (see figure 8)

```
> plot(result_no_cofactors, result_1, result_2, chr = c(1, 11, 15), col = c("black", "red", "b
+     lwd = c(3, 2, 1))
```

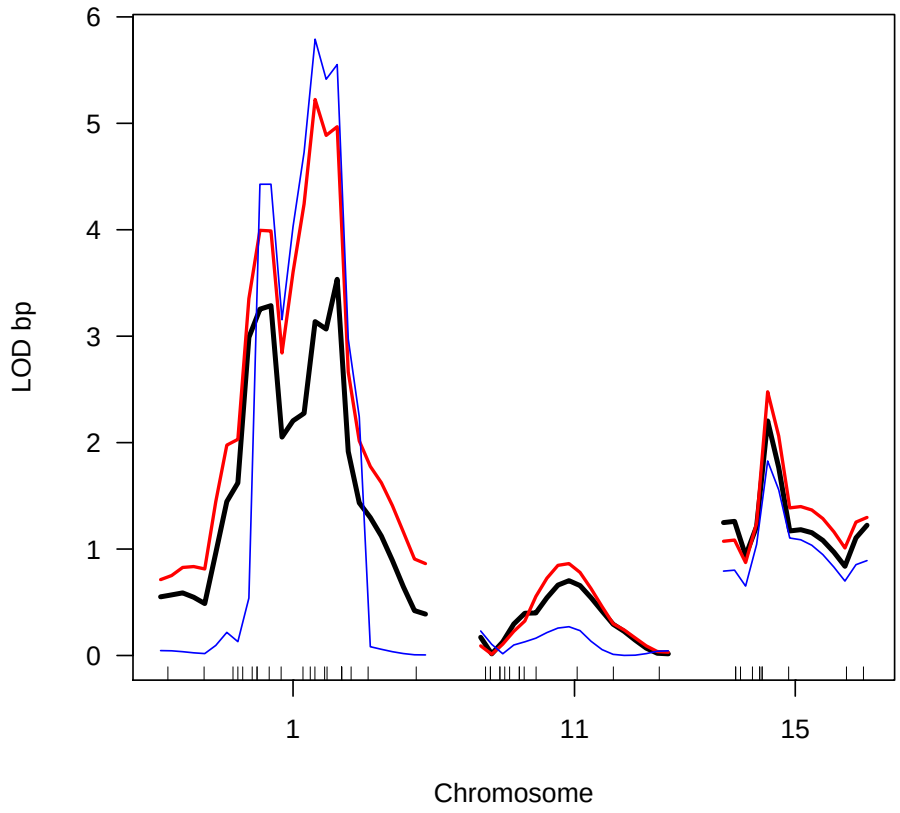

Figure 8: Three way comparison of the last scans. We see here a closeup of chromosome 1 scanned using *MQM* with 3 different cofactor settings: no cofactors (black), a single cofactor on chromosome 4 (Red) and cofactors on chromosome 1 and 4.(Blue) The LOD scores on chromosome 1 increase with more cofactors, while the peaks on the chromosome 11 and 15 decrease

5 Unsupervised backward elimination

MQM provides unsupervised backward elimination on a large number of markers by setting cofactors beforehand. When `samplesize` allows it a a cofactor is set at every marker. However, normally only a subset of locations is of interest. The `cofactor` function creates a list consisting of multiple cofactors. Iteratively the algorithm analyses all the markers and drops the least informative from the model. This step is repeated until a limited number of significant cofactors is left.

After backward elimination *MQM* scans each chromosome using the model that includes the retained cofactors. Here we set a cofactor at every fifth marker using `mqmsetcofactors`, and assess which chromosomes may be implicated in high bloodpressure (results in figures 9 and 10):

Automatic cofactor selection through backward elimination:

```
> cofactorlist <- mqmsetcofactors(hyperaug, 5)
> result <- mqmscan(hyperaug, cofactorlist, plot = T)


> mqmgetmodel(result)

  QTL object containing imputed genotypes, with 1 imputations.

       name chr  pos n.gen
Q1  D1Mit19   1 37.2     2
Q2 D1Mit102   1 75.4     2
Q3  D4Mit81   4 31.7     2
Q4 D5Mit148   5 14.2     2
Q5 D5Mit213   5 60.1     2
```

The `mqmgetmodel` function returns the final model from the resulting `scanone` type object. This model can be used with the `scantwo` routine from R/qtl. `mqmgetmodel` can only be used after backward elimination as it requires a list of cofactors. The resulting model can also be used to obtain the location and name of the significant cofactors.

```
> op <- par(mfrow = c(2, 1))
> plot(mqmgetmodel(result))
> plot(result)
```

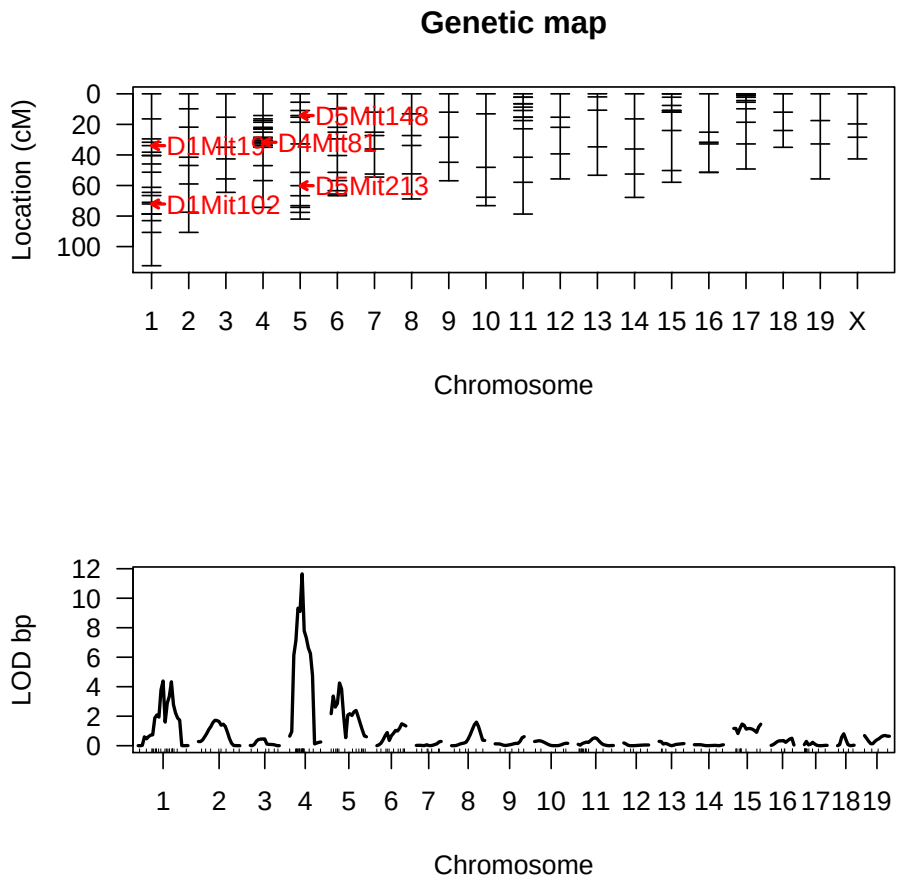

Figure 9: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice using *MQM*, after introducing cofactors at every fifth marker and backward elimination

Plot result of MQM backward elimination against that of `scanone`*:*

```
> plot(result, result_scanone, col = colors, lwd = lines)
```

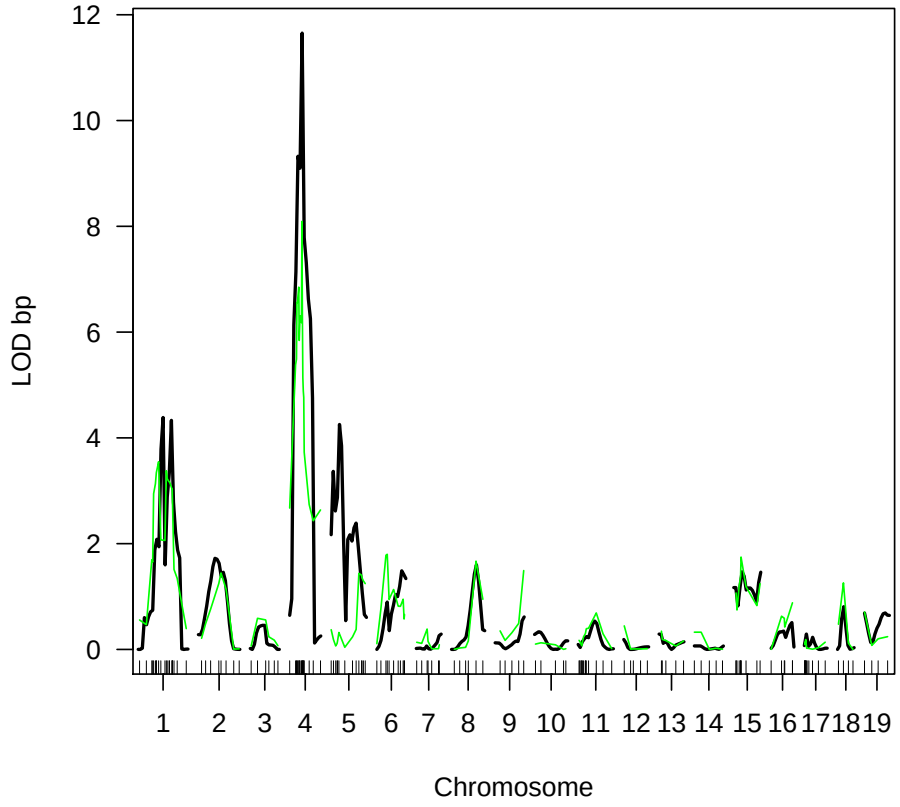

Figure 10: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice comparing *MQM* (black) and single QTL mapping with `scanone` (green)with cofactors at every fifth marker for *MQM* followed by backward elimination (`cofactor.significance = 0.02`)

MQM QTL mapping may result in many significant markers, with multiple hits on each chromosome. Figure 10 shows that at `cofactor.significance=0.02` chromosomes 1,2,4,5,6 and 15(?) are involved. Lowering the significance level from 0.02 to 0.002 may yield a smaller model. In biology extensive models are sometimes preferred, but in general a simpler model is more easily understood and, perhaps, validated. Increasing this significance level (hopefully) has the advantage that we can be more sure of the QTL.

Plot with lowered `cofactor.significance`*:*

```
> result <- mqmscan(hyperaug, cofactorlist, cofactor.significance = 0.002)
> op <- par(mfrow = c(2, 1))
> plot(mqmgetmodel(result))
> plot(result)
```

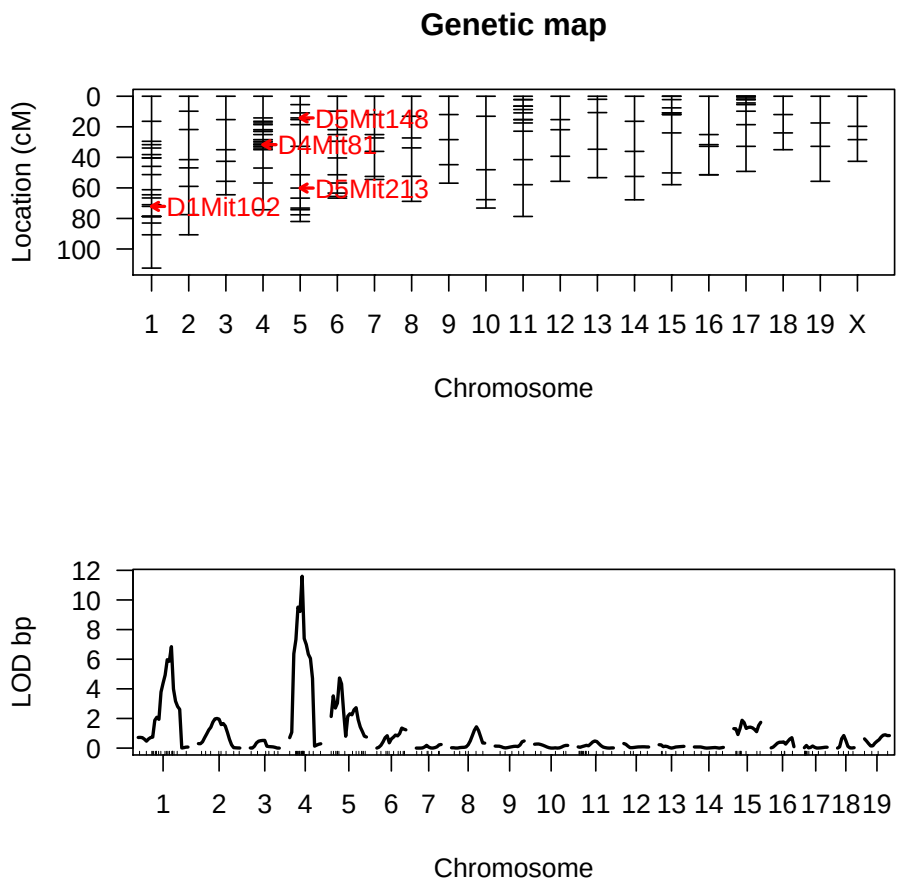

Figure 11: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice using *MQM*, initially with cofactors at every fifth marker; after backwards elimination (`cofactor.significance = 0.002`).

When comparing the *MQM* scan in figure 11 with the original `scanone` result in figure 4 there are some notable differences. Some QTL show higher significance (LOD scores) and some others show lower significance and are, therefore, estimated to be less likely involved in this trait.

Figures can be reconstructed from the result (`scanone` object) using the `mqmplot_one` function (see, for example, figure 12). Here the model and QTL profile are retrieved. These functions can only be used with `mqmcan`, as they have an additional column `mqm` which contains the QTL model. This extra column/list also contains the *estimated* information content per marker.

The column `mqm` is calculated from the deviation of the *ideal* marker distribution. For example, with a dataset of 100 mice, when comparing two distinct phenotypes at a marker location, we have most power when both groups are equally divided into 50 mice, and, virtually, no power when we have one mouse versus a group of 49 mice. We can multiply the estimated QTL effect by this information content to 'clean' the QTL profile by giving less weight to less informative markers. Please note that in the sample size already plays a role in calculating QTL. Meanwhile it allows (informal) further weighting/exploring *information* content (figure 12).

6 MQM effect plots

Effects can be add to plots using `mqmplot_directedqtl`, which requires setting `outputmarkers=TRUE` with `mqmscan` (default). To remove plotting the virtual markers introduced by *MQM* use `mqmextractmarkers(result)`. The plotting function adds the sign of the effect to the LOD score of the marker and calls the standard R/qtl plotting functions. An example can be found in figure 13.

Earlier figure 11 implied that chromosomes 1, 2, 4 and 5 are associated with high blood pressure. If we want to investigate the effect we can use *R/qtl*'s standard plotting tools to visualize *main* and/or *epistatic* effects. The following plots show these for markers "D1Mit102" (main effect, figure 14) and the interaction between "D1Mit102" and "D5Mit213" (figure 15). These markers are selected chosen for significance, based on mathematical distributions used by *MQM* (a better method, based on permutations, is discussed in section 7).

The initial scans for high bloodpressure in figure 9 possibly show two QTL on chromosome 1. The dual humped shape might be explained by an interaction between two QTL. We can find out if there is an interaction on chromosome 1 by using the `effectplot` function. To investigate the possible interaction we select markers "D1Mit19" (significant in figure 9) and "D1Mit102" (significant in figure 10 and 12). See figure 15.

Likewise, in case we are interested in the interactions between chromosome 1 and 5, we could make interaction plots between the two markers with a high LOD score on those chromosomes by using the effect plot. See figure 16.

Figure 15 shows some possible evidence for an interaction between the two markers D1Mit102 and D5Mit213, as both lines are non parallel. The two loci 'x' and 'y' influence the trait in a significant way (because they were selected as cofactors by *MQM*). If the effect shows two parallel lines the effect of both loci would be independent of one another. I.e. there is no significant interaction between x and y. Interactions like these are prime wetlab candidates, and, perhaps, tell us something about the biology behind the QTL.

```
> mqmplot_one(result, extended = TRUE)
```

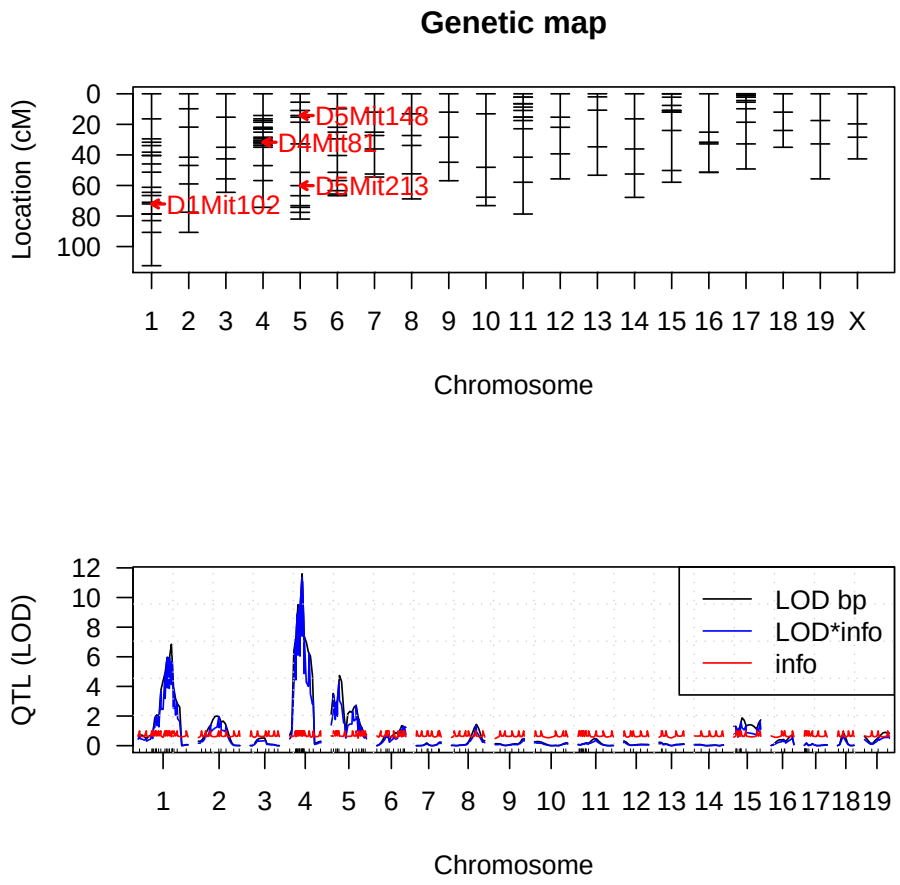

Figure 12: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice using *MQM*, initially with cofactors at every fifth marker; after backward elimination (`cofactor.significance = 0.002`). `mqmplot_one` function shows retained information in `result` with added weighting of information at marker positions

```
> dirresults <- mqmplot_directedqtl(hyperaug, result)
```

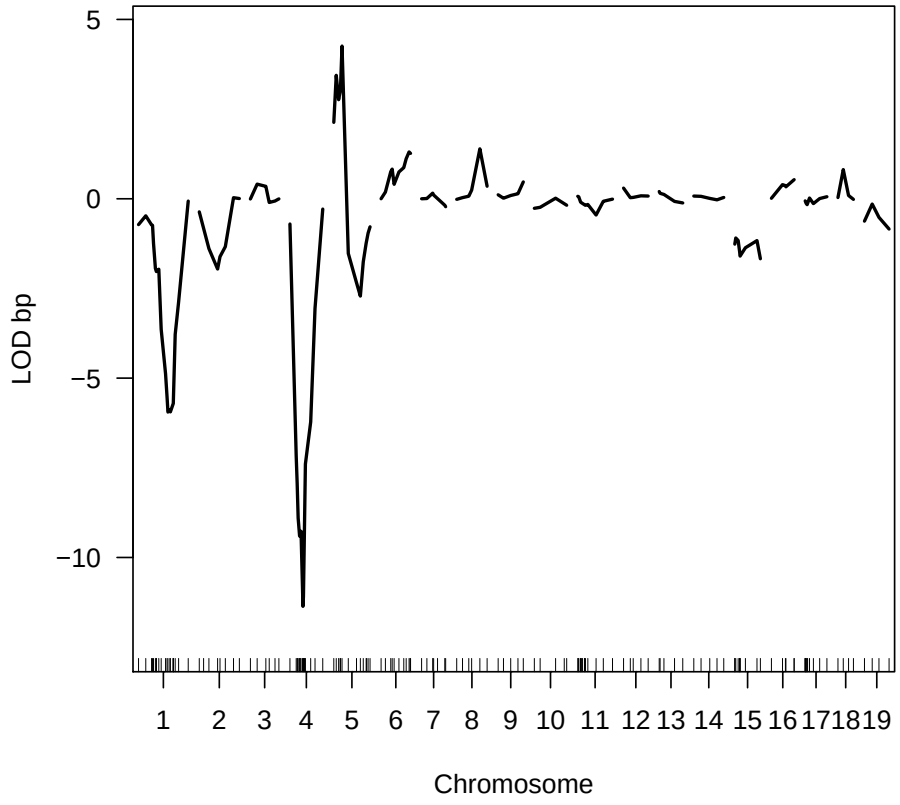

Figure 13: Same as figure 12, but with effect sign added to the QTL profile.

```
> plot.pxg(hyperaug, marker = "D1Mit102")
```

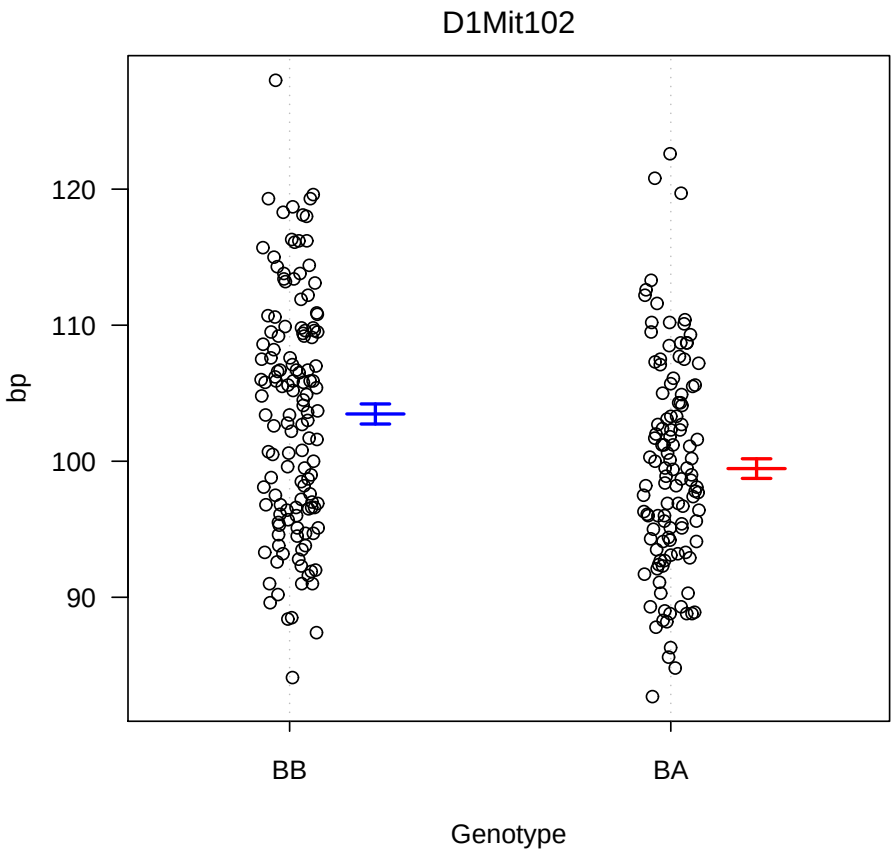

Figure 14: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice using *MQM*. Show effect of marker D1Mit102 using `plot.pgx`

```
> effectplot(hyperaug, mname1 = "D1Mit19", mname2 = "D1Mit102")
```

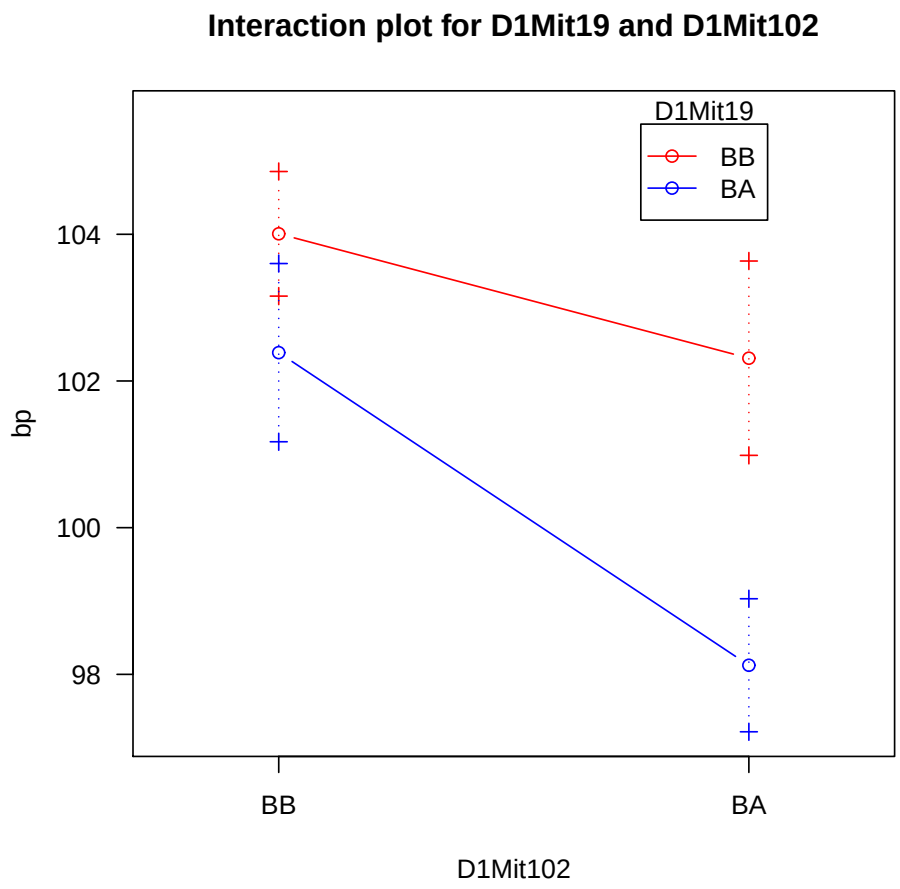

Figure 15: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice using *MQM*. `effectplot` shows possible small epistatic effects between markers D1Mit19 and D1Mit102

```
> effectplot(hyperaug, mname1 = "D1Mit102", mname2 = "D5Mit213")
```

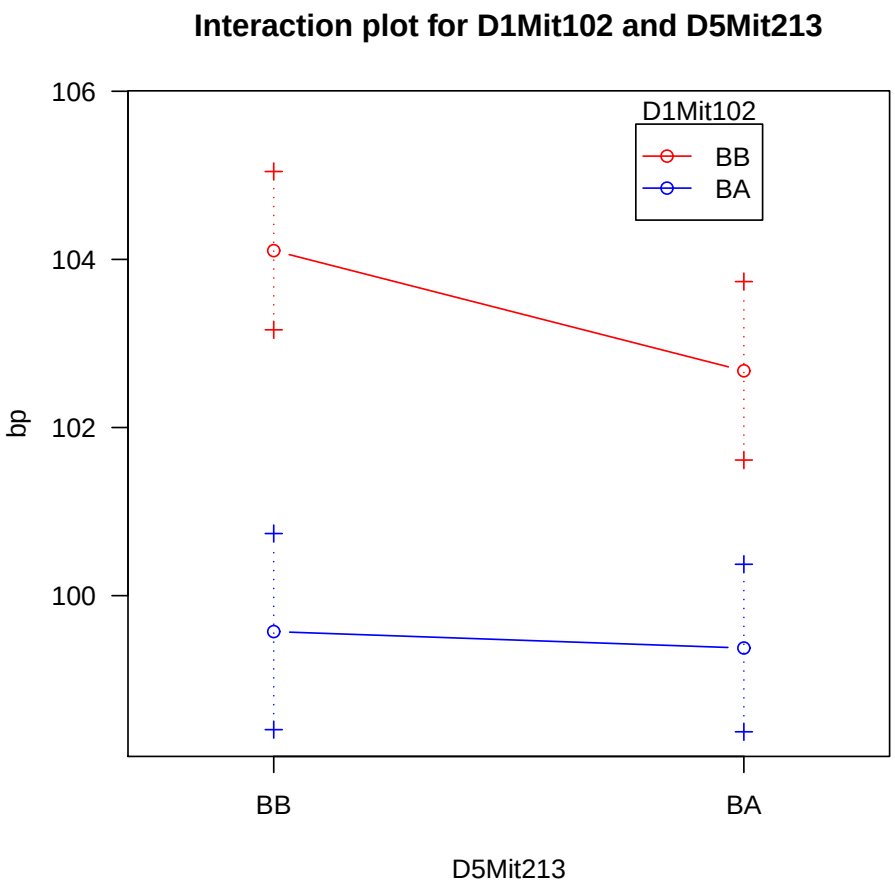

Figure 16: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice using *MQM*. `effectplot` shows an epistatic effect between markers D1Mit102 and D5Mit213

7 QTL significance

To estimate significance of QTL (and perhaps further exclude markers from a model) permutation testing is provided by the function `mqmpermute`. This step is computationally expensive[5] as the same test is repeated many times on shuffled data. Each test calculates LOD scores for non associated (randomly ordered) data.

MQM provides parametric and non-parametric bootstrapping to estimate QTL significance. The `bootmethod` parameter of the `mqmpermute` function selects the type. If you have access to multiple CPUs on your computer you can use the SNOW package [6, 5], which allows parallel computations on multiple CPU/cores. The snowpackage is available on cran, using Rgui it can be installed by selecting from the menu, Packages and choosing Install Package(s) from the dropdown menu. Select a cran mirror near your own location for fast download and then select the SNOW package. It will start downloading the package, and install any dependencies needed. Linux users can download a copy of SNOW from `http://cran.r-project.org/web/packages/snow/index.html`. Once the package has finished downloading the tar.gz file can be installed using R CMD INSTALL snow.tar.gz

Calculate significance - using SNOW parallelization parameters:

```
> require(snow)
> results <- mqmpermute(hyperaug, cofactors = cofactorlist, n.clusters = 2, n.run = 25, b.size

> resultsrqtl <- mqmprocesspermutation(results)
> summary(resultsrqtl)

LOD thresholds (25 permutations)
    [,1] [,2] [,3]
5%  2.86 2.86 2.86
10% 2.13 2.13 2.13
```

For small datasets, with a limited amount of classical traits, `mqmpermute` works fine. For large genome wide association studies (GWAS) use `mqmscanfdr` instead, which estimates false discovery rates (FDR) across the entire dataset at LOD cutoff. The routines have similar parameters.

To estimate FDR, whole genome information is permuted with `mqmscanfdr`. This method takes correlation between traits into account and gives an unbiased estimate of FDR at different (user specified) thresholds. The function scans the traits and counts observed QTL (markers with a LOD above `x`) when setting a certain threshold. It permutes all the data leaving the correlation structure between traits intact. For this small example very high FDR estimates are calculated because of the small amount of permutations and the high correlation between traits. We thus discover many QTLs that map to the same location, this could normally only happen when we have an information sparse marker, or correlated traits, often seen in microarray experiments.

Calculate FDR:

```
> data(multitrait)
> multifilled <- fill.geno(multitrait)
> mqmscanfdr(multifilled, mqmscanall, cofactors = cofactorlist, n.clusters = 2)
```

[5] In the tutorial, for all examples, 25 permutations are used. A real experiment should use over 1000 permutation tests

```
> mqmplot_permutations(results)
```

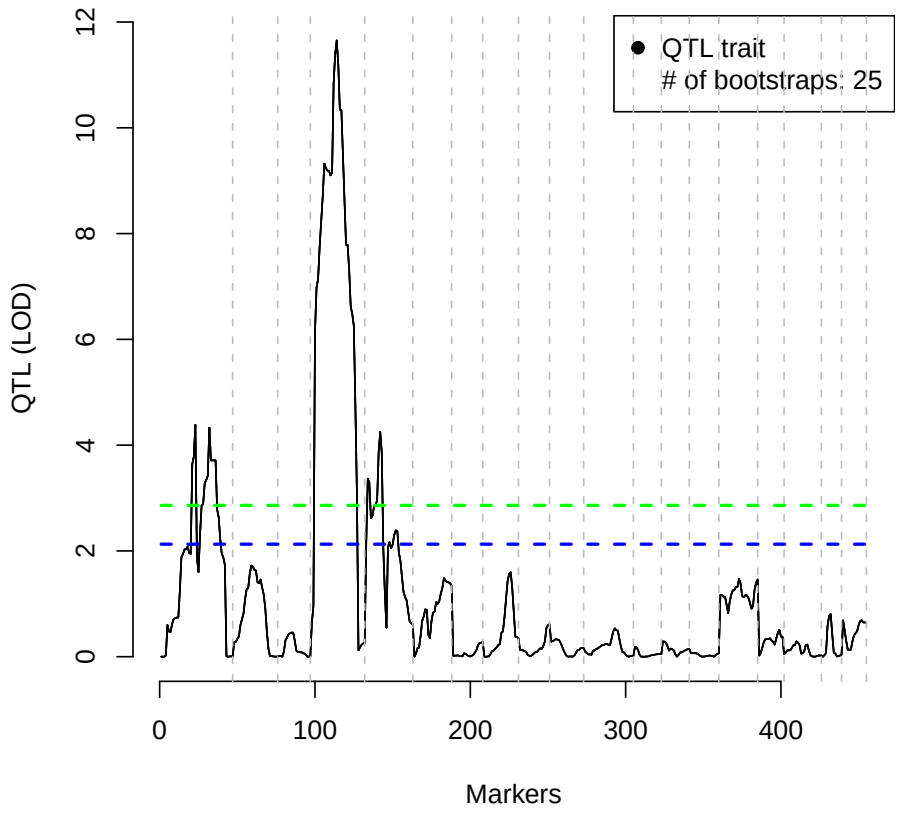

Figure 17: QTL profiles of the hyper dataset with the trait bp (blood pressure) in an experiment with 250 mice using *MQM*. Calculate significance permuting 25 times over QTL with a LOD higher than 2.5 LOD score can be considered significant (at `cofactor.significance=0.05` (green) or `cofactor.significance=0.10` (blue)). Estimation from permuting a single trait. Chromosomes are signified by the gray grid lines.

```
Calculation of FDR estimate of threshold in multitrait analysis.
QTL's above threshold: 107
Starting permutation 1
Starting permutation 2
Starting permutation 3
Starting permutation 4
Starting permutation 5
Starting permutation 6
Starting permutation 7
Starting permutation 8
Starting permutation 9
Starting permutation 10
   above.in.real.res above.in.perm.res
```

```
1              1246            791.3 0.6350722
2               732            473.6 0.6469945
3               547            368.9 0.6744059
4               430            281.7 0.6551163
5               355            222.2 0.6259155
7               272            142.1 0.5224265
10              204             83.9 0.4112745
15              136             37.8 0.2779412
20              107             21.8 0.2037383
```

The function `mqmpermute` does single trait permutations, and does not take correlation between the traits into account. The advantage is that a confidence interval is provided for each significant QTL. The *MQM* output needs to be converted to the standard R/qtl format using the `mqmprocesspermutation` function. The resulting object is of class `scanoneperm` and can be used by the standard R/qtl functions for further analysis.

Finally, with regard to parallelized processing using the multiple cores, nowadays even standard in laptop computers, we can assign, for example four cores using a batch size of 10. This way each core gets 10 traits to calculate QTL profiles on. When QTL modeling and mapping is done for these 10 traits. Another 10 traits are send to the core, until all traits have been analyzed. Using larger batches is efficient as for every batch the full R environment is started and initialized.

8 Parallelized xQTL analysis

MQM can analyse, so called, xQTL traits simultaneously using parallel computing on multiple CPU/cores, and even computer clusters. xQTL datasets (expression eQTL, metabolite mQTL) usually contain a large amount of phenotypes with known locations on the genome. These locations can be used for detecting cis/trans regulation, for example. For QTL mapping every phenotype requires one or more calls to `mqmscan`. In addition special plots are presented for xQTL studies.

As an example, the mQTL dataset `multitrait`, an *Arabidopsis thaliana* RIL cross, contains 24 metabolites measured (phenotypes). Of these 24 phenotypes we will only scan the first 5 phenotypes by setting the `pheno.col` parameter. To map back the regulatory locations of these metabolites one can use plain scanning of all metabolites (initially without cofactors). Next, we plot all the profiles in a heatmap (see figure 18). In this heatmap the colors represent the LOD score, on the x-axis the marker number and on the y-axis the metabolite. The traits are numbered in the plot. Plot heatmap without cofactors and then the heatmap with cofactors and backward elimination. Figure 19 shows improvement over figure 18 because of an improved signal to noise ratio.

```
> data(multitrait)
> multifilled <- fill.geno(multitrait)
> resall <- mqmscan(multifilled, pheno.col = c(1, 2, 3, 4, 5), n.clusters = 2)


> cofactorlist <- mqmsetcofactors(multifilled, 3)
> resall <- mqmscan(multifilled, pheno.col = c(1, 2, 3, 4, 5), cofactors = cofactorlist, n.clu
```

Use `mqmplot_nice` for more graphical output. (Unfortunately this does not show in the generated PDF, but in R it shows the trait profiles)

```
> mqmplot_multitrait(resall, "I")
```

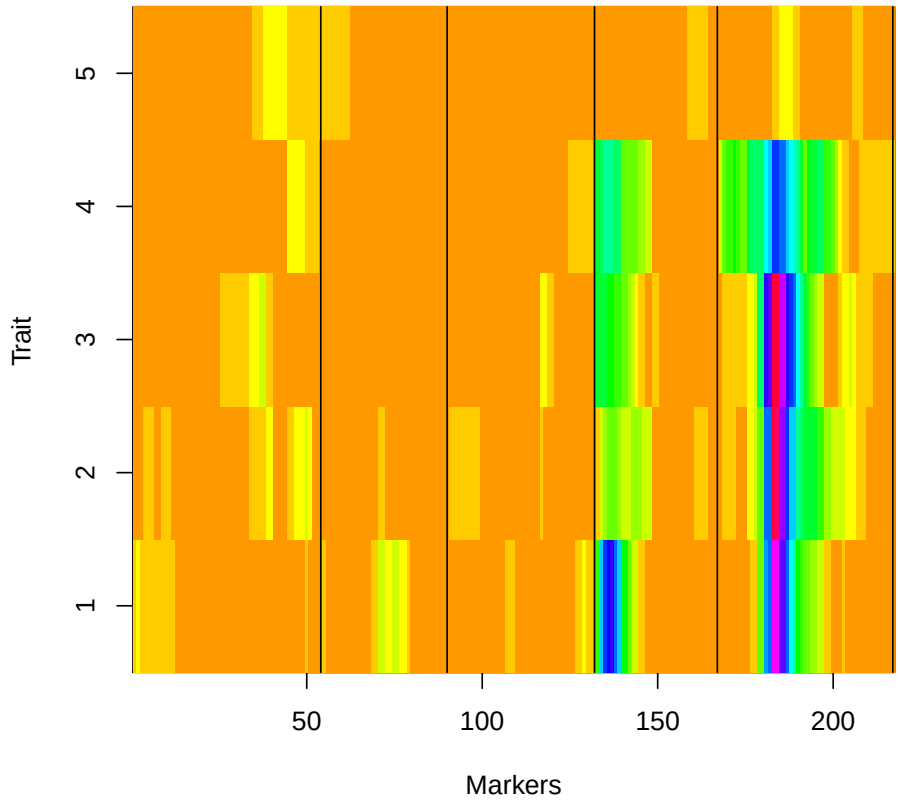

Figure 18: *Arabidopsis thaliana* RIL mQTL dataset with 24 metabolites as phenotypes. Heatmap of metabolite expression traits, with profiles created using *MQM* without preselected cofactors. The colors represent the LOD score, on the x-axis the marker number and on the y-axis the metabolite

```
> mqmplot_nice(resall, legendloc = 1)
```

The next plot is the `mqmplot_circle`. It shows a circular representation of the genome. After using automatic backward selection certain markers are found to be significant using *MQM*. When highlighting, the size of the markers is scaled, based on the LOD score of the marker for the highlighted trait. (**FIXME!**)Improve description of this plot (**FIXME!**)Multiple traits show QTL outside the chromosome (solid colored squares)

The next plot is `mqmplot_cistrans`. This plot is only available when genomic locations of the traits are known, typically in xQTL studies. By default the R/qtl cross object does not store this data. So the user has to add this information to the cross object using the addloctocross function. After this operation the cis_trans plot can be created for QTL with associated genome locations. For example with expression QTL (eQTL) usually the probes on the microarray have a known chromosomal location.

The two axis of the cistrans plot both show the genetic location. The x-axis is, normally, the QTL location and the y-axis the locations of the trait (e.g. a the microarray probe).

```
> mqmplot_multitrait(resall, "I")
```

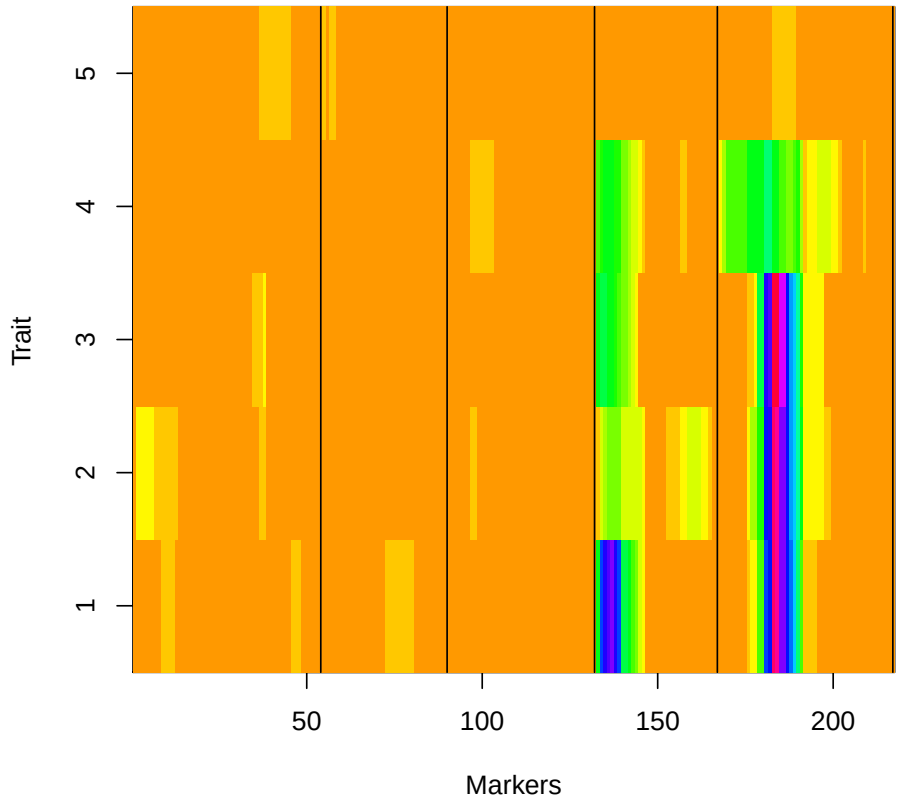

Figure 19: *Arabidopsis thaliana* RIL mQTL dataset with 24 metabolites as phenotypes. Heatmap of metabolite expression traits, with profiles created using *MQM* with cofactors at each third marker. The colors represent the LOD score, on the x-axis the marker number and on the y-axis the metabolite

When having locations we can, again, use the `mqmplot_circle` function, now with the extra information.

```
> mqmplot_circle(multifilled, resall)
```

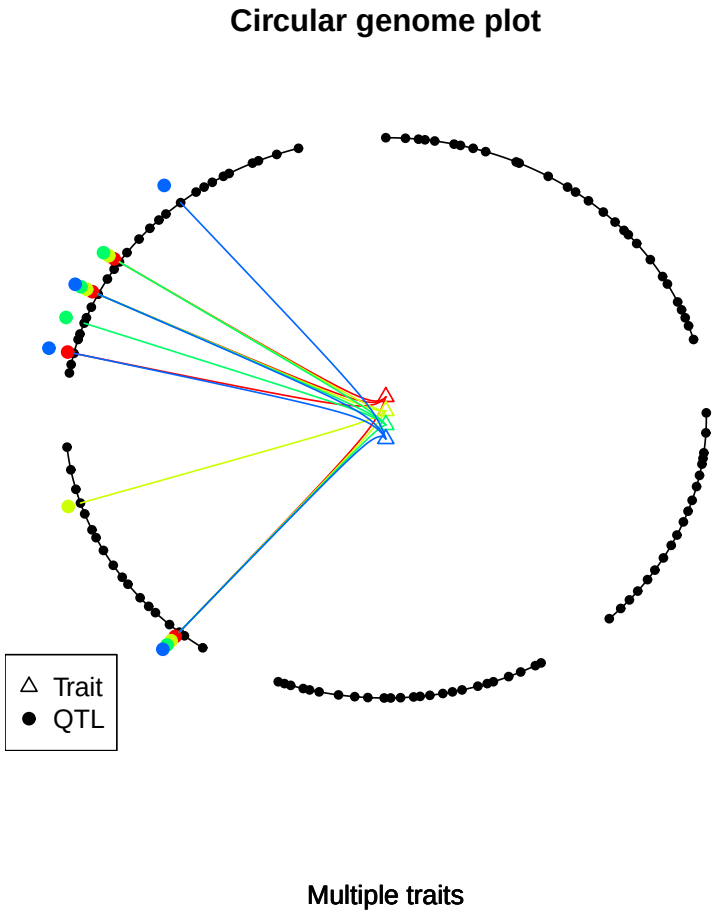

Figure 20: Circleplot 1 - Multiple traits without locations, traits are in the centre connected by a colored spline to their QTL locations

```
> mqmplot_circle(multifilled, resall, highlight = 2)
```

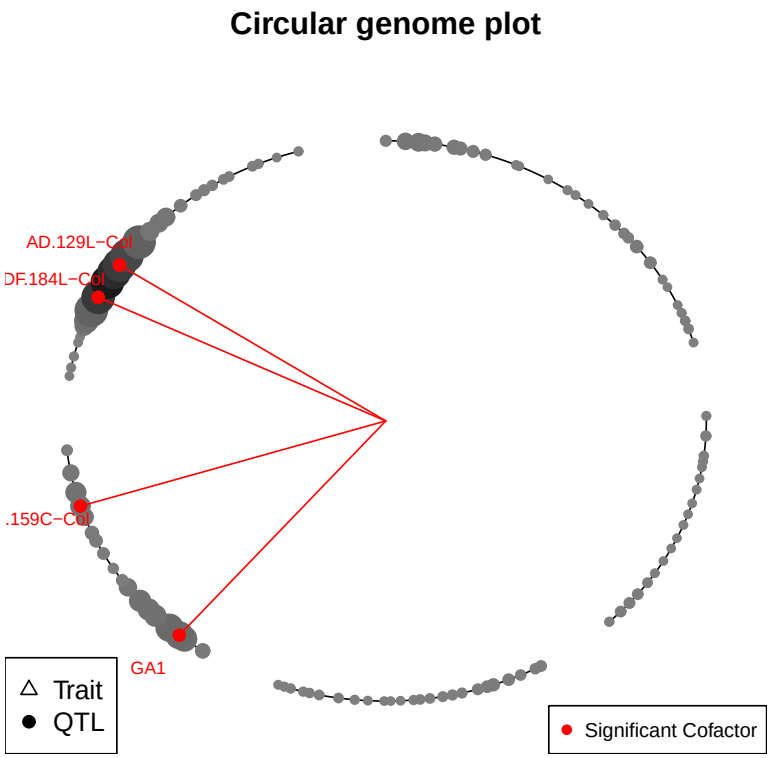

Figure 21: Circleplot 2 - Multiple traits without locations with a highlight on trait 2

```
> data(locations)
> multiloc <- addloctocross(multifilled, locations)

Phenotypes in cross: 24
Phenotypes in file: 24

> mqmplot_cistrans(resall, multiloc, 5, FALSE, TRUE)

Total maplength: 500  cM in  5 Chromosomes
The lengths are: 0 130 215 300 385
```

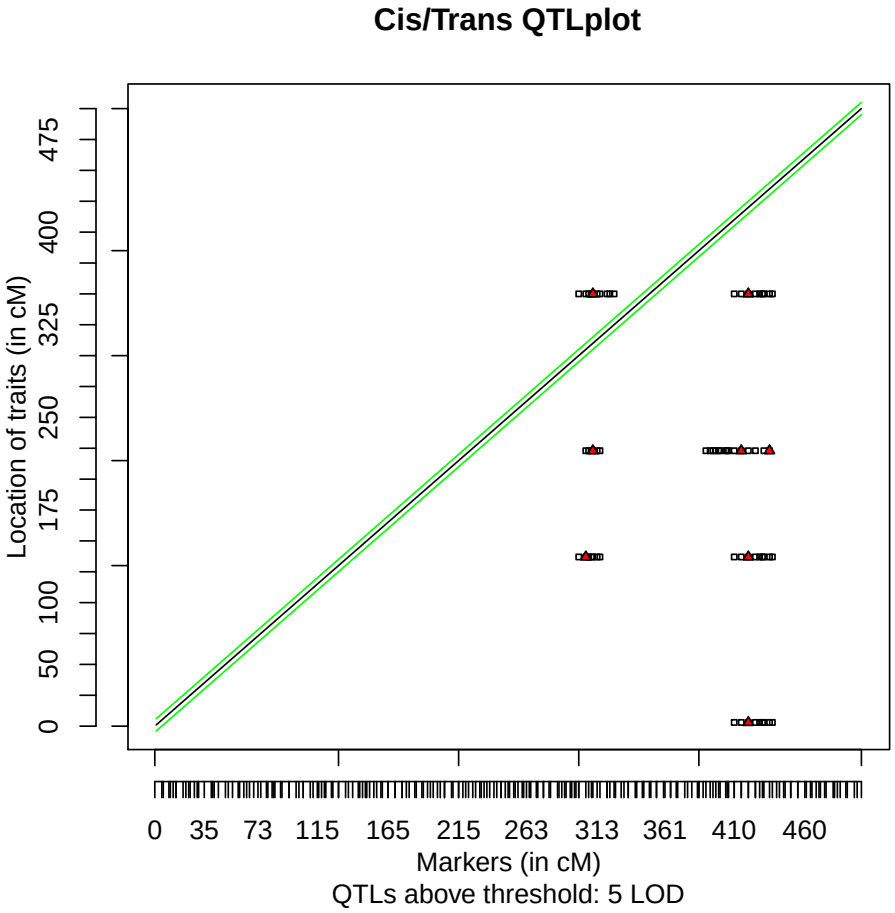

Figure 22: *Arabidopsis thaliana* RIL mQTL dataset with 24 metabolites as phenotypes. mqmplot_cistrans, available when QTL have associated genome locations. QTL are plotted against the position on the genome they were measured (here mQTL for *Arabidopsis thaliana*), cutoff is at a lod score of 5

```
> mqmplot_circle(multiloc, resall, highlight = 2)
```

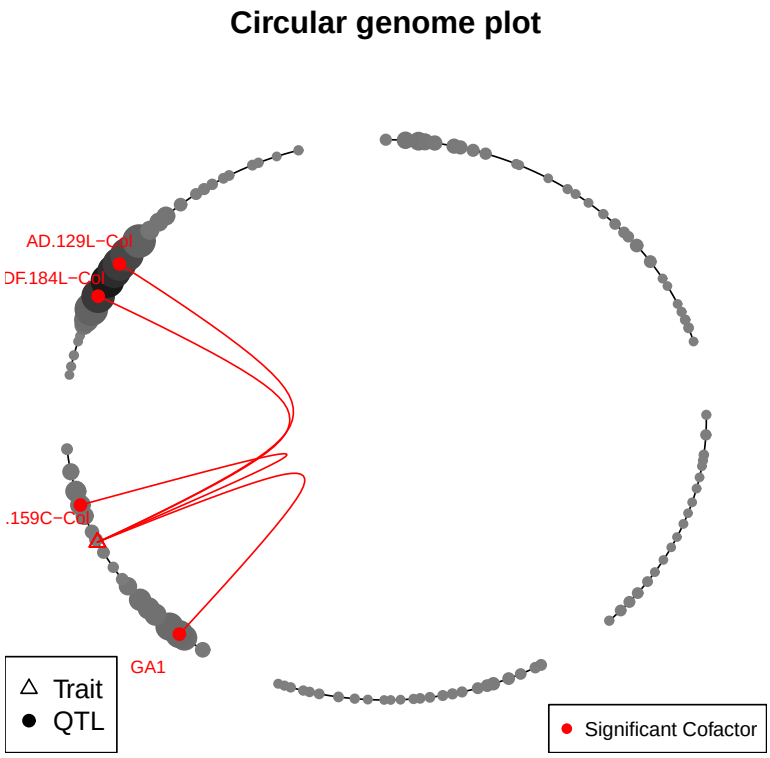

Figure 23: *Arabidopsis thaliana* RIL mQTL dataset with 24 metabolites as phenotypes. Circleplot 3 - Multiple traits with locations added, here we highlight trait number 2

9 Overview of all *MQM* functions

Table 1: Added functionality

mqmaugment:	mqm data augmentation
mqmscan:	mqm modeling and scanning
mqmsetcofactors:	Set cofactors at these markers (or every x marker)
which.marker:	Change markernumbering into mqmformat
mqmscanall:	mqmscanall to scan all traits using mqm
mqmpermute:	Single trait permutation
mqmscanfdr:	Genome wide False Discovery Rates
mqmprocesspermutation:	Creates an R/qtl permutationobject from the output of the `mqmpermute` function
mqmplot_multitrait:	plotting of multiple traits (MQMmulti object)
mqmplot_nice:	plotting of mutiple traits (MQMmulti object)
mqmplot_directedqtl:	Plotting of signle trait with added qtl effect
mqmplot_boot:	plot methode to show single trait permutations
mqmplot_one:	plotting of single trait analysis with information content
mqmplot_cistrans:	Genomewide plot of cis- and transQTLs above a threshold
addloctocross:	Adding genetic locations for traits
mqmtestnormal:	Tests the normallity of a trait

(`NOTE: `): the bibliography should use standard bibtex

References

[1] Broman, K.W.; 2009. *A brief tour of R/qtl* http://www.rqtl.org, R/qtl tutorials.

[2] Karl W. Bromand and Saunak Sen. *A Guide to QTL Mapping with R/qtl* Springer, 2009

[3] Broman, K.W.; Wu, H.; Sen, S.; Churchill, G.A.; 2003. *R/qtl: QTL mapping in experimental crosses* Bioinformatics, 19:889-890.

[4] Jansen R. C.; 2007. *Chapter 18 - Quantitative trait loci in inbred lines* Handbook of Stat. Genetics 3th edition,(c) 2007 John Wiley & Sons, Ltd.

[5] Tierney, L.; Rossini, A.; Li, N.; and Sevcikova, H.; 2004. *The snow Package: Simple Network of Workstations* Version 0.2-1.

[6] Rossini, A.; Tierney, L.; and Li, N.; 2003. *Simple parallel statistical computing* R. UW Biostatistics working paper series University of Washington. 193

[7] Jansen R. C.; Nap J.P.; 2001 *Genetical genomics: the added value from segregation* Trends in Genetics, 17, 388-391.

[8] Jansen R. C.; Stam P.; 1994 *High resolution of quantitative traits into multiple loci via interval mapping* Genetics, 136, 1447-1455.

[9] Jansen R.C. *Controlling the Type I and Type II Errors in Mapping Quantitative Trait Loci.* Genetics, Vol 138, 871-881

[10] Churchill, G. A.; and Doerge, R. W.; 1994 *Empirical threshold values for quantitative trait mapping* Genetics 138, 963-971.

[11] Jansen R. C.; 1993 *Interval mapping of multiple quantitative trait loci* Genetics, 135, 205-211.

[12] Dempster, A. P.; Laird, N. M. and Rubin, D. B.; 1977 *Maximum likelihood from incomplete data via the EM algorithm* J. Roy. Statist. Soc. B, 39, 1-38.

[13] Zeng, Z. B.; 1993 *Theoretical basis for separation of multiple linked gene effects in mapping quantitative trait loci* Proc. Natl. Acad. Sci. USA, 90, 10972-10976.

[14] Zeng, Z. B.; 1994 *Precision mapping of quantitative trait loci.* Genetics, 136, 1457-1468