A Tutorial On R Package "QTLRel" (V1.1 or Later)

Riyan Cheng

School of Medicine
University of California, San Diego
Email: rcheng1207@gmail.com

June 15, 2022

Contents

1 Introduction

Advanced intercross lines (AILs) are an ideal resource for fine-scale mapping of quantitative trait loci (QTL). Unlike populations such as F_2, individuals in an AIL population are genetically unequally related. The unequal relatedness among individuals requires appropriate statistical models and computational tools. We have developed a computationally efficient R package 'QTLRel' for analysis of quantitative AIL data. Users can use the package to calculate condense identity coefficients (CIC) as well as kinship coefficients from large pedigrees, estimate variance-covariance component parameters, perform a genome-scan for QTL, assess genome-wide significance (e.g., gene dropping) and plot results.

Though initially designed for analysis of AIL data, **almost all of the major functions have a very general usage**. For instance, the function `scanOne` can be used to perform a genome scan for nuclear family data using linear mixed models (LMM) where relatedness (or population structure) is a concern as long as relationship matrices are available.

To help users to understand and use the package 'QTLRel', we provide this tutorial with concrete examples that illustrate the process of data analyses using the package. The examples are not isolated; earlier examples may also be a preparation for later ones. We recommend inexperienced R users study the entire tutorial to write your own program. However, this tutorial should not substitute for the R documentation; instead, it should serve as supplemental material.

Codes in this tutorial can be easily adapted for custom data.

2 Install and Load Package

We can install the package from within R using the command

```
> install.packages("QTLRel",dependencies=TRUE)
```

and load the package using the command

```
> library(QTLRel)
```

Package source and documentation are available on CRAN (here). We may need gdata for re-ordering so we load it as well.

```
> library(gdata)
```

3 Data Formats

In this tutorial, we'll use part of our real data in F_8, including a pedigree 'pedF8', phenotype data 'pdatF8', genotype data 'gdatF8' and a genetic map 'gmapF8'. The sample size is

approximately 500.

The **pedigree** should be a data frame that includes individual ID 'id', father 'sire' and mother 'dam'. Other information such as sex and generation is optional. If given, 'sex' should be "M", "Male" or 1 for a male and "F", "Female" or 2 (other than 0 and 1) for a female, and 'generation' can be numeric 0, 1, 2, ... or non-numeric "F0", "F1", "F2", ..., which should be in an increasing order. The special symbol 0 is reserved for unknown IDs. If a sire/dam is an **inbred** founder, its ID should be tagged by character 'i'. The following are a few lines in the pedigree 'pedF8'.

```
> data(miscEx)
> pedF8[c(1:3,131:133),]
       id sex generation sire dam family
1       1   M         F1   1i  2i     F1
2       2   F         F1   1i  2i     F1
3   16152   F         F2    1   2  F1-16
131 18278   F         F2    1   2  F1-25
132 18279   F         F2    1   2  F1-25
133 18280   F         F2    1   2  F1-25
>
```

The sire with ID "1i" and dam with ID "2i" indicate that they are inbred. To obtain the same kinship coefficients as well as condense identify coefficients, we can also use the following pedigree.

```
> pedF8.1[1:5,]
     id sex generation sire dam family
1     1   0         F1    0   0     F1
2 16152   F         F2    1   1  F1-16
3 16153   F         F2    1   1  F1-16
4 16154   F         F2    1   1  F1-16
5 16149   M         F2    1   1  F1-16
```

Note that F_2's are created by selfing one F_1 individual so as to ensure the same identify coefficients as obtained from a pedigree that starts with two inbred lines, which in turn have to be created by many generations of brother-sister mating (or selfing). As a problem, this pedigree can't be used to simulate X-chromosome (e.g. in gene dropping). Alternatively, we can add 30–50 generations of selfing (or 100–150 generations of brother-sister mating) to create inbred lines (see pedF8.2 for this pedigree with 150 generations of brother-sister mating). The following results show that the above pedigrees lead to basically identical kinship coefficients.

```
> ksp<- kinship(pedF8, ids=rownames(pdatF8), all=TRUE, msg=FALSE)
> ksp.1<- kinship(pedF8.1, ids=rownames(pdatF8), all=TRUE, msg=FALSE)
> ksp.2<- kinship(pedF8.2, ids=rownames(pdatF8), all=TRUE, msg=FALSE)
>
> dim(ksp)
[1] 500 500
>
> sum(abs(ksp-ksp.1))
[1] 0
> sum(abs(ksp-ksp.2))
[1] 6.869755e-10
> sum(abs(ksp.1-ksp.2))
[1] 6.869755e-10
```

The **genotype data** should be a matrix or a data frame, with each row representing an observation and each column a marker locus. The row names (optional) can be individual IDs and the column names should be marker names. The following are five lines in the genotype data 'gdatF8'. Genotypes 1, 2 and 3 correspond to "AA", "AB" and "BB".

```
> gdatF8[1:5,1:5]
      rs6269442 rs13475701 rs13475706 rs3716083 rs3722996
33461         3          3          3         3         3
33615         2          2          2         2         2
33649         3          3          3         3         3
35608         1          1          1         1         2
31805         2          2          2         2         2
>
```

The **phenotype data** 'pdatF8' has four columns: 'sex', 'age', 'bwt' and 'cage'. The variable 'bwt' is the body weight of an individual, 'age' is the age (in days) when body weight was measured, and 'cage' is the cage where the individual was bred. The following are five lines in the phenotype data 'pdatF8'. Note that there were a total of 163 cages.

```
> pdatF8[1:5,]
      sex age  bwt     cage
33461   F  70 18.6 BDF8-147
33615   M  81 24.2 BDF8-133
33649   F  80 16.0 BDF8-141
35608   F  64 18.6   BDF8-4
31805   M  76 26.2  BDF8-71
```

```
>
> length(unique(pdatF8$cage))
[1] 160
>
```

The **genetic map** should be a data frame with colums 'snp', 'chr', 'dist', where 'snp' is the SNP (marker) name, 'chr' is the chromosome where the 'snp' is located, and 'dist' is the genetic distance in centi-Morgan (cM) on the chromosome. In 'gmapF8', the extra column 'phyPos' is the physical location (in bp; build 37) of the SNP on the chromosome.

```
> gmapF8[815:819,]
           snp chr   dist phyPos37
815  rs3725637  18 18.692 34588212
816 rs13483299  18 19.461 36279727
817  rs3664831  18 19.461 37453885
818  rs3699816  18 21.088 39473386
819 rs13483319  18 21.395 41067707
>
```

The above data comes with R package QTLRel and can be loaded with the following command

```
> data(miscEx)
```

4 Statistical Models

Consider the following statistical model (see Cheng et al (2011) for the general model which this package can deal with)

$$y_i = \boldsymbol{x}_i'\boldsymbol{\beta} + x_i^* a^* + z_i^* d^* + u_i + \epsilon_i, \quad i = 1, 2, \cdots, n \tag{1}$$

where y_i is the trait value for the i-th individual, $\boldsymbol{x}_i$ represents covariates (e.g. sex) and $\boldsymbol{\beta}$ are the corresponding effects, x_i^* is 1, 0 or -1 if the genotype at the putative QTL is AA, Aa or aa and a^* is the additive effect of the putative QTL, z_i^* is 1 if the genotype at the putative QTL is heterozygous or 0 if the genotype is homozygous and d^* is the dominance effect, u_i represents polygenic variation, and ϵ_i denotes environmental effect. Assume that $\epsilon_i \sim N(0, \sigma^2), i = 1, 2, \cdots, n$ are independent, and $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)' \sim N_n(\boldsymbol{0}, \boldsymbol{G})$ with $\boldsymbol{G} = (g_{ij})$ and is independent of $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \cdots, \epsilon_n)'$. It is known that in general (e.g. Abney et al, 2000; Cheng et al, 2010)

$$g_{ij} \;\; = \;\; 2\Phi_{ij}\sigma_a^2 + \Delta_{ij,7}\sigma_d^2 + (4\Delta_{ij,1} + \Delta_{ij,3} + \Delta_{ij,5})Cov(a, d)$$

$$
\begin{aligned}
& +\Delta_{ij,1}\sigma_h^2 + (\Delta_{ij,1} + \Delta_{ij,2} - f_i f_j)\mu_h^2 \\
= \; & g_{a,ij}\sigma_a^2 + g_{d,ij}\sigma_d^2 + g_{ad,ij}Cov(a,d) \\
& +g_{h,ij}\sigma_h^2 + g_{m,ij}\mu_h^2 \qquad (2)
\end{aligned}
$$

where Φ_{ij} is the kinship coefficient between the i-th and j-th individuals, f_i is the inbreeding coefficient for the i-th individual, and Δ_{ij}'s are condensed identity coefficients as defined in Lynch and Walsh (1998, pp.133) and can be calculated from the pedigree data.

5 Condensed Identity Coefficients

To fit model (1), we need identity coefficients in (2). Condensed identity coefficients (cic) can be calculated from the pedigree, using function `cic`.

`Usage:`

```
cic(ped,ids,inter,df=3,ask=FALSE,verbose=FALSE)
```

The 'ped' is the pedigree that should have at least three components 'id', 'sire' and 'dam'. Preferably, generation information is provided. Otherwise, the program will try to derive generation information based on the pedigree. The 'ids' specifies the IDs of the individuals for which to calculate the Jacquard condensed identity coefficients. If 'ids' is not specified, all individuals in the pedigree 'ped' will be considered. It is recommended that 'ids' be specified only for interested individuals because extra IDs may result in overwhelming computation.

Condensed identity coefficients can be derived from generalized kinship coefficients. Bottom-up and top-down are two strategies for calculating generalized kinship coefficients from a pedigree. The bottom-up approach starts from the target individuals and moves up till the founders. It requires minimal storage but the computational load can be approximately exponential in the number of generations. The bottom-up approach is not realistic if the number of generations is large and the number of individuals in each generation is not small. The top-down approach starts from founders and moves down to the target individuals. The computational load is approximately linear in the number of generations. However, the intermediate generalized kinship coefficients need to be stored, which may require extensive storage if the number of individuals in a generation is large. This function allows for a hybrid of both bottom-up and top-down approaches by specifying intermediate generations via the argument 'inter' such that generations of too many members are passed over. If 'inter' is missing, the function will use 'df' to determine the "optimal" configuration of 'inter'. The default value of 'inter' is 3. If 'df = 0', then there will be no intermediate generations, which results in implementation of the bottom-up approach. If 'df' is large, then all generations will be used as intermediate generations, which calls for the top-down approach.

We can set 'ask = TRUE' to see if 'inter' produced by the program is feasible. If not, we can tune 'df' until it is satisfactory. Setting 'verbose = TRUE' will print out some messages to track the running process.

The output of `cic` is a matrix G of nine columns with $G[, j]$ being the j-th Jacquard identity coefficients. Once we finish `cic`, we can run `genMatrix` to extract five genetic matrices, including additive genetic matrices "AA", dominance genetic matrix "DD" and other three that are described in Abney et al (2000), as well as inbreeding coefficients "ib".

Tips: You may need the **administrative privilege** to run this function on systems such as Windows 7 [1].

Example 1. Calculate condensed identity coefficients from a pedigree.

```
> # only interested individuals in F8
> id<- rownames(pdatF8)
> id[1:5]
[1] "33461" "33615" "33649" "35608" "31805"
> # check if phenotype and genotype data are from the same sample
> sum(!is.element(id,pedF8$id))
[1] 0
> sum(!is.element(id,rownames(gdatF8)))
[1] 0
> sum(!is.element(rownames(gdatF8),id))
[1] 0
>
> # running
> idcf<- cic(pedF8,ids=id,df=3,ask=TRUE,msg=TRUE)
   Suppose !0, !1, !M and !Male stands for female...
   ------------------------------------
     id sire dam sex generation old.id
381 381  266 227   M         F7  32889
   ------------------------------------
   Above should be dam(s)...

  Total free disk space needed: 593.629 Mb...
  Carry-over number of individuals in each generation:
-999   F1   F2   F3   F4   F5   F6   F7   F8
```

[1] If your system is Windows Vista or 7, you may need to start R as follows: 1) click "start" menu; 2) go to all programs; 3) go to R; 4) go the R version (e.g., R 2.9.2); 5) right click on it; 6) choose "Run as..." from the drop-down menu; 7) select "Administrator".

```
   1    3   73   65   45   37   54  145  500
  Will go over generations: -999F1F2F4F5F6F7F8
Continue? Yes/No: Yes
  This may take hours or days to finish!
  Please wait or press 'ctrl-c' to abort...
 F1 F2 F4 F5 F6 F7 F8
   Finishing.....................................................................
> # extract genetic matrices
> gmF8<- genMatrix(idcf); names(gmF8)
[1] "ib" "AA" "DD" "AD" "HH" "MH"
> dim(gmF8$AA)
[1] 500 500
> gmF8$AA[1:3,1:5]
         33461    33615    33649    35608    31805
33461 1.504395 1.026367 1.069580 1.038330 1.060547
33615 1.026367 1.504883 1.013184 1.013184 1.015381
33649 1.069580 1.013184 1.503906 1.066895 1.044922
```

Genetic matrices can also be estimated using genotypic data in the sense of identity-by-state (IBS). The function `genMatrix` has this capability in case there are only two founder strains or genotypes are only recorded as, say, "AA", "AB" and "BB" (QTLRel_0.2.7 or later).

6 Variance Components

Once we have genetic matrices, we can proceed to estimate variance components that are needed in a genome scan. Estimating variance component parameters can be achieved via function `estVC`.

```
Usage:
     estVC(y,x,v=list(E=diag(length(y))),initpar,nit=25,
        control=list(),hessian=FALSE)
```

where 'y' is a numeric vector or a numeric matrix of one column that represents a phenotype, and 'x', if not missing, is a vector representing a covariate or a data frame or matrix each column of which represents a covariate. 'v' is a list of interested variance components, such as additive and dominance genetic matrices, with **"E" denoting the residual matrix and always being included**. If interested in Hessian matrix, we can set 'hessian = TRUE'. Here is an example.

Example 2. Estimate variance components.

```
> idx<- !is.na(pdatF8[,"bwt"])
> pdatTmp<- pdatF8[idx,] # remove missing values
> gdatTmp<- gdatF8[match(rownames(pdatTmp),rownames(gdatF8)),] #matching
>
> ii<- match(rownames(pdatTmp),rownames(gmF8$AA))
> vc<- estVC(y = pdatTmp[,"bwt"], x = pdatTmp[,c("sex","age")],
+    v=list(AA=gmF8$AA[ii,ii],DD=gmF8$DD[ii,ii]))
> vc$value # likelihood
[1] -978.2822
> vc$par # parameter estimates
 (Intercept)         sexM          age           AA           DD            E
9.106219e+00 5.944458e+00 1.093796e-01 3.943530e+00 7.154003e+00 5.569893e-08
```

In the above example, the trait of interest is body weight 'bwt'. We include both additive and dominance genetic variance components but ignore the other three by setting `HH=NULL,AD=NULL` and `MH=NULL`. We also consider 'sex' and 'age' as covariates without questioning whether it makes sense to do so. Of course, we can test an effect as illustrated below.

Example 3. Include non-genetic variance components.

```
> ranMtr<- rem(~cage, data=pdatTmp) # matrice for random effect (cage)
> names(ranMtr)
[1] "cage"
> dim(ranMtr$cage)
[1] 496 496
> vc.cage<- estVC(y = pdatTmp[,"bwt"], x = pdatTmp[,c("sex","age")],
+    v=list(AA=gmF8$AA[ii,ii],DD=gmF8$DD[ii,ii], cage=ranMtr$cage))
>
> 2*(vc.cage$value - vc$value) # likelihood ratio test for cage effect
[1] 20.43798
```

The above example demonstrates how to include 'cage' as a non-genetic random effect, and also gives the likelihood ratio test statistic for 'cage' effect. However, we will ignore 'cage' effect as we move on.

7 Performing Genome Scans

Now we come to the point to perform a genome scan using function `scanOne`.

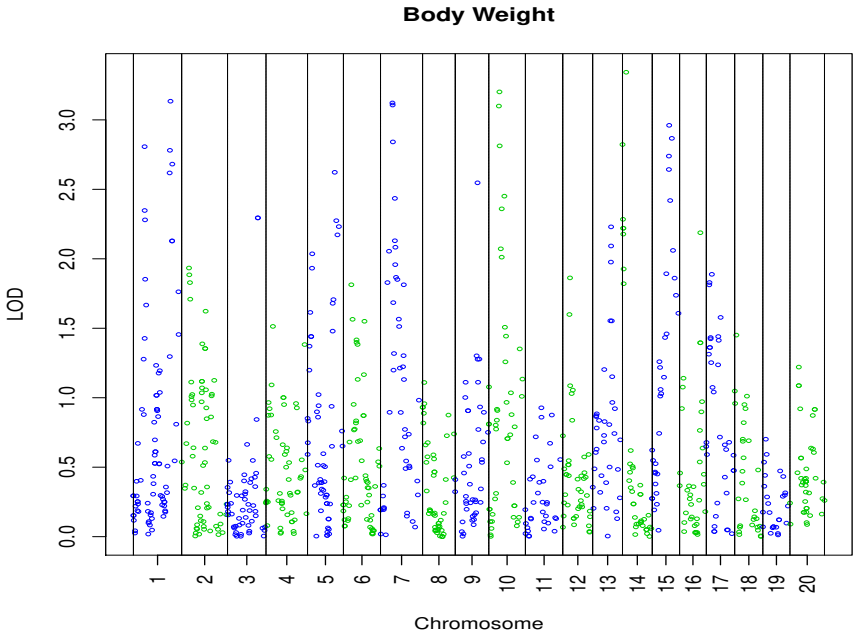

Figure 1 Sex and age are non-interactive covariates.

Usage:

```
scanOne(y, x, gdat, prdat = NULL, vc = NULL, intc = NULL,
   numGeno = FALSE, test = c("None","F","LRT"),
   minorGenoFreq = 0, rmv = TRUE)
```

The meanings of 'y' and 'x' are the same as in `estVC`. 'gdat' is genotype data in the format described previously. It is ignored if an object 'prdat' from `genoProb` is specified as an argument. 'vc' is an object from `estVC` or `aicVC`, or an estimated variance-covariance matrix induced by relatedness and environment. 'intc', if provided, specifies covariates that interact with QTL. Note that if we treat numerical coding of genotype as numerical value, we should set 'numGeno = TRUE'.

The function will return a list with at least the following components:

1) F or LRT: the F-test or likelihood ratio test (LRT) statistic at the SNP (marker) if 'test' is "F" or otherwise

2) pval: P-value at the snp (marker) if 'test' is "F" or "LRT"

3) parameters: estimated parameters at all scanning loci, including additive effect 'a' and dominance effect 'd' if 'prdat' is not "NULL".

If the genome scan is based on genotype data 'gdat', then we need make sure there are not missing genotypes. Otherwise, we can call function `genoImpute` to impute missing genotypes.

```
Usage:
     genoImpute(gdat,gmap,prd=NULL,step=Inf,gr=2,pos=NULL,
        method=c("Haldane","Kosambi"),na.str="NA",verbose=FALSE)
```

where 'gdat' is genotype data, 'gmap' is genetic map, and 'prd' is an object from `genoProb` if not "NULL". If we only impute missing genotypes at marker loci, we can specify 'step = Inf' (by default). The argument 'gr' indicates which generation is under consideration. The missing genotype is randomly assigned with a probability conditional on the genotypes of the flanking makers. Currently, this function only works for AIL. Here is an example.

Example 4. Genome scan without interactive covariates.

```
> sum(is.na(gdatTmp)) # number of missing genotypes
[1] 140
> gdatTmpImputed<- genoImpute(gdatTmp,gmapF8,gr=8,na.str=NA)


>
> # genome scan
> lrt<- scanOne(y=pdatTmp[,"bwt"], x=pdatTmp[,c("sex","age")],
+     gdat=gdatTmpImputed, vc=vc)
>
> plot(lrt,gmap=gmapF8,main="Body Weight") # plotting
>
```

In the above example, we first call `genoImpute` to impute 142 missing genotypes and then perform a genome scan. The scan takes both sex and age as non-interactive covariates. We can also include interactive covariates as illustrated in the following example where age is a non-interactive covariate but sex is an interactive covariate. Figure 1 is the Manhattan plot of the mapping results.

Example 5. Genome scan with an interactive covariate.

```
> # genome scan: interactive sex
> lrt.sex<- scanOne(y=pdatTmp[,"bwt"], x=pdatTmp[,c("age")],
+     gdat=gdatTmpImputed, intc=pdatTmp[,c("sex")],vc=vc)
>
> plot(lrt.sex,gmap=gmapF8,main="Body Weight") # plotting
>
```

Then QTL by sex interaction effect can be easily obtained as follows.

Example 6. Extract test statistics.

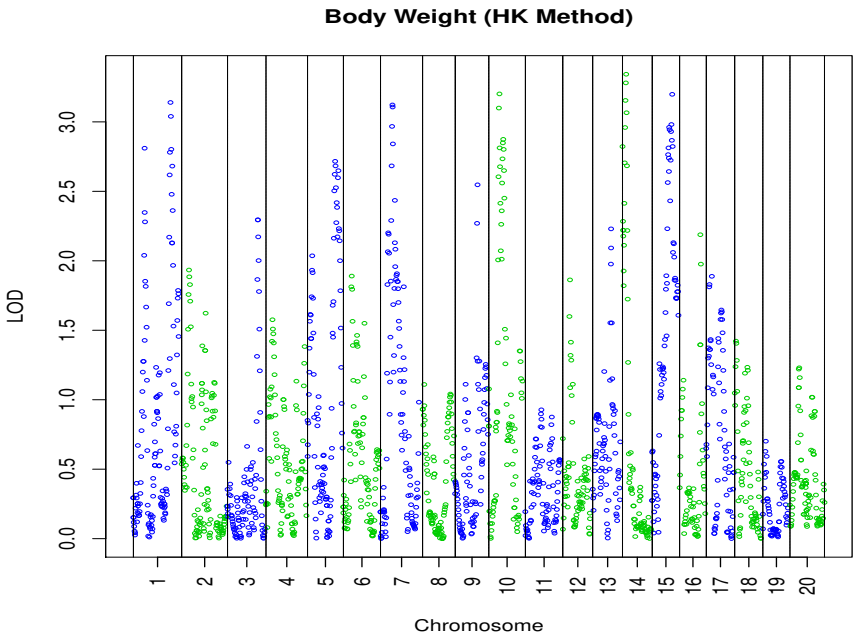

Figure 2 Sex and age are non-interactive covariates.

```
> # QTL by sex interaction
> lrtTmp<- lrt
>    lrtTmp$LRT<- lrt.sex$LRT - lrt$LRT
>
> plot(lrtTmp,gmap=gmapF8,main="Body Weight: QTL by Sex Effect") # plotting
>
```

In case of AIL mapping populations, the package 'QTLRel' can also perform Haley-Knott interval mapping. This is achieved by first calling `genoProb` to calculate genotype probabilities and calling `scanOne` to scan the genome.

Usage:

```
genoProb(gdat,gmap,step=Inf,gr=2,pos=NULL,method=c("Haldane",
   "Kosambi"),verbose = FALSE)
```

where 'gdat' is genotype data whose entry should be 1, 2, 3 or 0, corresponding to "AA", "AB", "BB" or missing genotype, and 'gmap' is a genetic map. The argument 'step' defines scanning loci such that the "historical" genetic distance between any two adjacent loci for which probabilities are calculated is not larger than 'step'. If 'step = Inf' (or large enough), probabilities will only be calculated at loci in both the columns of 'gdat' and the rows of 'gmap'. If 'step' is small, a large set of putative loci will be considered, including all loci

defined by the columns of 'gdat' and the rows of 'gmap'. The output is a list with the following components:

1) pr: a 3-D array with the first dimension equal to that of 'gdat', the second corresponding to three genotype and the third to the putative loci. The probabilities will be -1 if not imputable, which happens when the genotype data is missing at all loci on the chromosome.

2) chr: chromosome where the locus is located.

3) dist: genetic distance (in cM) of the locus from the first locus on the chromosome.

4) snp: SNP (marker) that the locus represents.

Again, it is currently only suitable for advanced intercross lines. Let's look at an example.

Example 7. (Haley-Knott method) Genome scan without interactive covariates.

```
> gdTmp<- gdatTmp
>    gdTmp[is.na(gdTmp)]<- 0
>    unique(c(as.matrix(gdTmp)))
[1] 3 2 1 0
> prDat<- genoProb(gdat=gdTmp,gmap=gmapF8,step=3,gr=8)
>
> # genome scan: Haley-Knott method
> lrtHK<- scanOne(y=pdatTmp[,"bwt"], x=pdatTmp[,c("sex","age")],
+     prd=prDat, vc=vc)
>
> plot(lrtHK,main="Body Weight (HK Method)") # plotting
>
```

In the above example, we define 'step = 3', which results in 1741 scanning loci including the 847 markers. Also note that we don't need genetic map information in plotting because the information is contained in the object "lrtHK". Figure 2 displays the mapping results. The following is another example using Haley-Knott interval mapping.

Example 8. (Haley-Knott method) Genome scan with sex being an interactive covariate.

```
> # genome scan: Haley-Knott method, interactive sex
> lrtHK.sex<- scanOne(y=pdatTmp[,"bwt"], x=pdatTmp[,c("age")],
+     prd=prDat, intc=pdatTmp[,c("sex")],vc=vc)
>
> plot(lrtHK.sex,main="Body Weight (HK Method)") # plotting
```

```
>
> # Haley-Knott method, QTL by sex interaction
> lrtHKTmp<- lrtHK
>    lrtHKTmp$LRT<- lrtHK.sex$LRT - lrtHK$LRT
>
> plot(lrtHKTmp,main="Body Weight (HK Method): QTL by Sex Effect")
>
```

8 Empirical Thresholds

Empirical thresholds can be obtained by permutation. This simply repeats the genome scan with genotype data being shuffled among individuals. The function `nullSim` in the package can be called to estimate thresholds. However, we may write our own code to gain a better control. First, let's look at two permutation examples.

Example 9. Permutation test using marker genotype data.[2]

```
> nn<- nrow(gdatTmpImputed) # sample size
> ntimes<- 1000 # number of simulations
> cvMtr<- NULL # matrix to save results of permuted data
> for(n in 1:ntimes){
+    idx<- sample(1:nn,replace=FALSE) # permutation
+    tmp<- scanOne(y=pdatTmp[,"bwt"],x=pdatTmp[,c("sex","age")],
+        gdat=gdatTmpImputed[idx,], vc=vc)
+    cvMtr<- rbind(cvMtr,tmp$LRT)
+    cat(n,"/",ntimes,"\r") # track process
+ }
```

In the above example, we shuffle the order of observations and store the shuffled order in variable "idx" so that "gdatTmpImputed[idx,]" is the permuted genotype data from "gdatTmpImputed". The code for genome scan is the same otherwise. The results are stored in the matrix "cvMtr". This process is repeated "ntimes = 1000" times. The 0.05 genome-wide significance threshold can be estimated by

```
> quantile(apply(cvMtr,1,max),0.95) # in LRT
     95%
18.76991
```

[2]The permutation test have been regarded as a "gold" standard to assess statistical significance. The above approach, however, is computationally intensive. We and others have shown that naive permutation tests provide a great approximation and thus can be considered in practice.

```
> quantile(apply(cvMtr,1,max),0.95)/(2*log(10)) # in LOD
    95%
4.134378
```

Example 10. Permutation test in Haley-Knott interval mapping.

```
> # for HK method...
> cvMtrHK<- NULL
> for(n in 1:ntimes){
+    idx<- sample(1:nn,replace=FALSE)
+    prdTmp<- prDat
+    prdTmp$pr<- prdTmp$pr[idx,,]
+    tmp<- scanOne(y=pdatTmp[,"bwt"], x=pdatTmp[,c("sex","age")],
+        prd=prdTmp, vc=vc)
+    cvMtrHK<- rbind(cvMtrHK,tmp$LRT)
+    cat(n,"/",ntimes,"\r") # track process
+ }
```

Since we have a pedigree here, we can perform gene dropping test. In the following example, we first call `genoSim`, which does gene dropping, to generate genotype data set "gdatTmp", and then perform the genome scan using "gdatTmp". This example can be easily adapted to the Haley-Knott version.

Example 11. Gene dropping test.

```
> pedR<- pedRecode(pedF8) # recode the pedigree
> ids<- rownames(gdatTmpImputed) # relevant individual IDs
> cvMtrGD<- NULL
> for(n in 1:ntimes){
+    gdatTmp<- genoSim(pedR, gmapF8, ids=ids)
+    tmp<- scanOne(y=pdatTmp[,"bwt"],x=pdatTmp[,c("sex","age")],
+        gdat=gdatTmp, vc=vc)
+    cvMtrGD<- rbind(cvMtrGD,tmp$LRT)
+    cat(n,"/",ntimes,"\r")
+ }
```

9 Plotting

Object of `scanOne` can be plotted using R function `plot`, as illustrated in example 4 and 7. Here is one more example, in which we feed a threshold with 'cv' and tune the symbol size with 'cex'.

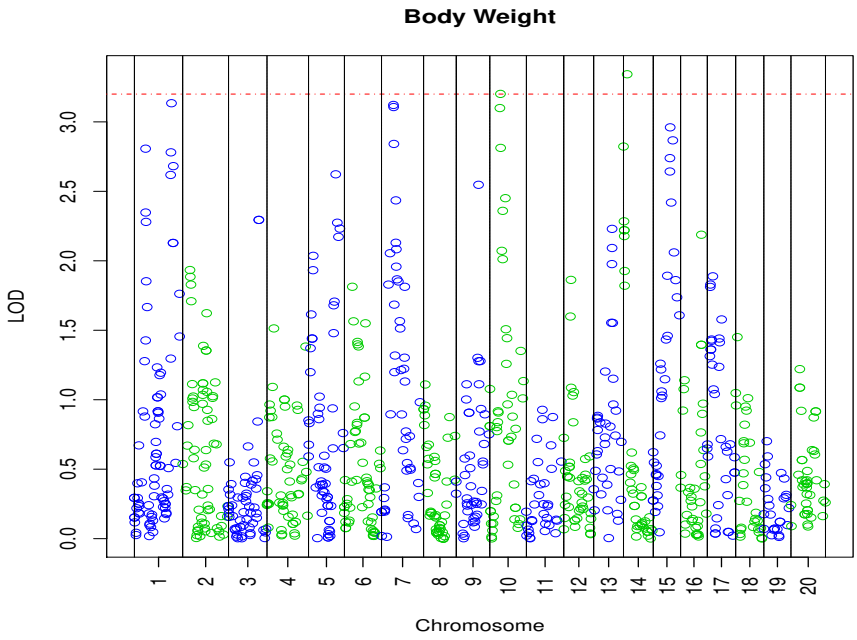

Figure 3 Plotting `scanOne` objects. Sex and age are non-interactive covariates.

Example 12. Plotting `scanOne` objects (Figure 3).

```
> plot(lrt,cv=3.2,gmap=gmapF8,main="Body Weight (HK Method)",cex=1)
```

We can gain more control using function `plotit`. Here are two examples of calling function `plotit`. Users may check the documentation (here) for its usage.

Example 13. Plotting mapping results using function `plotit` (Figure 4).

```
> idx<- match(colnames(gdatTmpImputed),gmapF8$snp)
> Tmp<- data.frame(chr=gmapF8$chr[idx],
+                  dist=gmapF8$dist[idx],
+                  y=lrt$LRT)
>    Tmp<- Tmp[order(Tmp$chr,Tmp$dist),] # order by chromosome and distance
>
> plotit(Tmp, cv=15, main="Mapping Plot of Body Weight", xlab="Chromosome",
+    ylab="LRT", col=as.integer(Tmp$ch)%%2+2,type="p",lty=2)
```

Example 14. Plotting mapping results by chromosome using function `plotit` (Figure 5).

```
> library(lattice) # dependency package
>
```

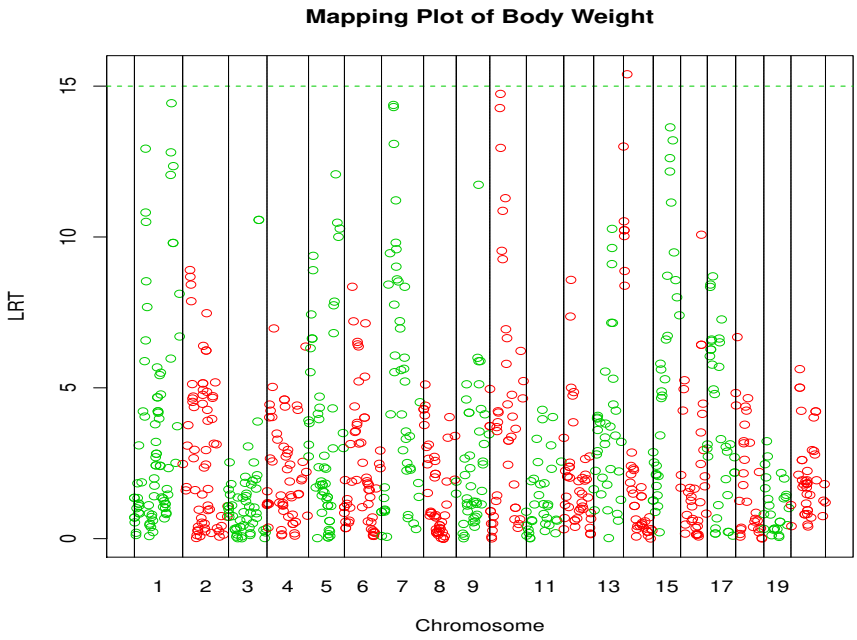

Figure 4 Plotting mapping results using function `plotit`. Sex and age are non-interactive covariates.

```
> tmp<- plotit(Tmp,cv=12,type="p",lty=2,col=4,cex=0.65,
+       xlab="Genetic Position (cM)",ylab="LRT",
+       main="Body Weight",bychr=TRUE)
> print(tmp)
```

10 Miscellaneous

In the package, there are a few other functions that may be useful.

- `kinship` calculates kinship coefficients from a pedigree.
- `mAIC` performs multiple QTL model section via AIC criterion. This function allows additive covariates but not interactive covariates.
- `lodci` calculates LOD support intervals.

  Example 15. LOD support intervals based on Haley-Knott interval mapping.

  ```
  > Tmp<- data.frame(chr=lrtHK$chr,
  +                  dist=lrtHK$dist,
  +                  y=lrtHK$LRT/(2*log(10))) # convert to LOD
  ```

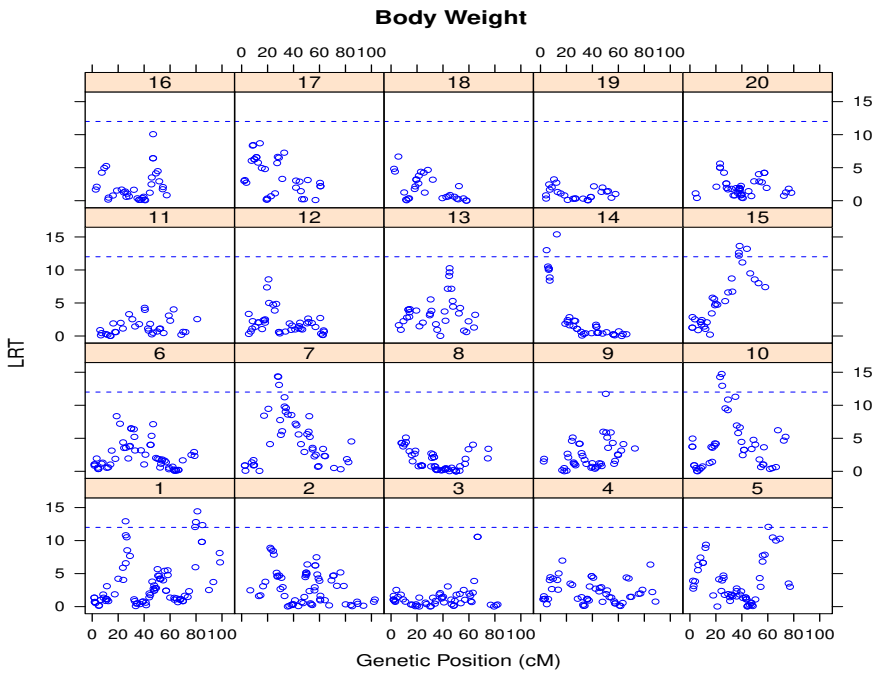

Figure 5 Plotting mapping results by chromosome using function `plotit`. Sex and age are non-interactive covariates.

```
>    Tmp$chr<- reorder(Tmp$chr)
>    Tmp<- Tmp[order(Tmp$chr,Tmp$dist),]
> lc<- lodci(Tmp,cv=3.2,lod=1.5,drop=1.5)
> lc
  chr    lower  upper index
1  10 20.82247 36.082   956
2  14  7.08000 15.830  1261
```

- `gls` can estimate QTL effects and standard errors by fitting a multiple QTL model, or perform significance tests. The following example show how to fit a model including two putative QTL at the 674-th and 1365-th scanning loci.

  Example 16. Estimated QTL effects and standard errors.

```
> # multiple QTL model estimates
> dtfTmp<- data.frame(
+    y=pdatTmp[,"bwt"],
+    age=pdatTmp[,"age"],
+    sex=pdatTmp[,"sex"],
+    a1=prDat$pr[,1,lc$index[1]]-prDat$pr[,3,lc$index[1]],
```

```
+    d1=prDat$pr[,2,lc$index[1]],
+    a2=prDat$pr[,1,lc$index[2]]-prDat$pr[,3,lc$index[2]],
+    d2=prDat$pr[,2,lc$index[2]]
+ )
> est1<- gls(y~age+sex+a1+d1+a2+d2,data=dtfTmp,vc=vc)
> est2<- gls(y~age+sex*(a1+d1+a2+d2),data=dtfTmp,vc=vc)
> est1 # with age and sex being additive covariates
              Estimate Std. Error   t value      Pr(>|t|)
(Intercept)  9.5185452 2.29047260  4.155712  3.829545e-05
age          0.1058518 0.01601744  6.608535  1.018670e-10
sexM         5.8911437 0.14344713 41.068398 1.843534e-160
a1           0.4781028 0.12872596  3.714114  2.274702e-04
d1           0.1831207 0.15539526  1.178419  2.392039e-01
a2          -0.4548940 0.11694440 -3.889831  1.142236e-04
d2          -0.2820369 0.15433786 -1.827399  6.825028e-02
> est2 # with sex being an interactive covariate
                Estimate Std. Error      t value     Pr(>|t|)
(Intercept)  9.483656508 2.27997325  4.159547277 3.773070e-05
age          0.105219167 0.01593714  6.602137851 1.067844e-10
sexM         5.953009844 0.25492814 23.351717356 2.475143e-81
a1           0.186598226 0.16324452  1.143059674 2.535789e-01
d1           0.220511463 0.21146193  1.042795105 2.975638e-01
a2          -0.545499464 0.15585031 -3.500150049 5.079906e-04
d2          -0.224303662 0.20958065 -1.070249870 2.850403e-01
sexM:a1      0.609847484 0.20964349  2.908974159 3.792978e-03
sexM:d1     -0.001905152 0.29002083 -0.006569016 9.947614e-01
sexM:a2      0.172029731 0.19550303  0.879933842 3.793318e-01
sexM:d2     -0.127489530 0.29337596 -0.434560248 6.640753e-01
>
```

- `qtlVar` calculates variance induced by QTL in a quantitative trait.

  Example 17. QTL induced variation based on Haley-Knott interval mapping.

```
> ii<- match(rownames(pdatTmp),rownames(gmF8$AA))
> vc0<- estVC(y = pdatTmp[,"bwt"], v=list(AA=gmF8$AA[ii,ii], DD=gmF8$DD[ii,ii]))
> nb<- length(vc0$par) - length(vc0$v)
> nr<- nrow(vc0$y)
> cov<- matrix(0,nrow=nr,ncol=nr)
```

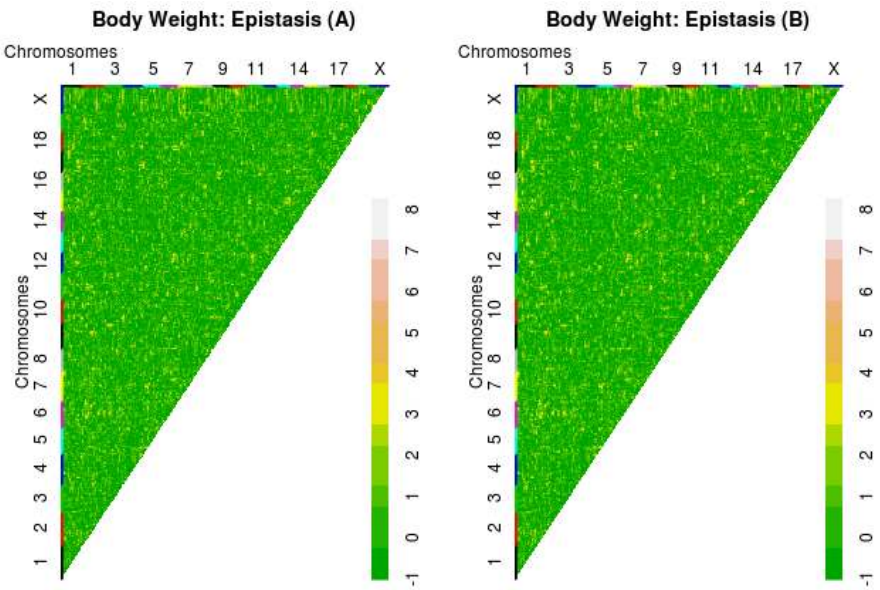

Figure 6 Plotting `scanTwo` objects. Sex and age are non-interactive covariates.

```
> for(i in 1:length(vc0$v))
+    cov<- cov + vc0$v[[i]]*vc0$par[nb+i]
> tv<- mean(diag(cov)) # total variation
> eff<- NULL # QTL effects
> for(n in 1:length(lrtHK$par)){
+    eff<- rbind(eff,lrtHK$par[[n]][c("a","d")])
+ }
> eff<- data.frame(eff) # data frame!
> qv<- qtlVar(eff,prDat$pr) # per QTL variation
> qv[lc$index]/tv*100 # per QTL heritability
[1] 0.7474018 0.7317993
```

The last command calculates variation percentage in body weight that are associated with two highest peaks in figure 2

- `scanTwo` calculates QTL-by-QTL interaction. Sometimes, people are interested in epistasis. The package provides this facility though it can be computationally expensive if the number of scanning loci is large. In the following example we calculate epistatic effect with sex and age being additive covariates and plot the results.

  Example 18. Calculating epistasis and plotting.

```
> qqInt.1<- scanTwo(y=pdatTmp[,"bwt"], x=pdatTmp[,c("sex","age")],
+     gdat=gdatTmpImputed, vc=vc)
>
> qqInt.2<- scanTwo(y=pdatTmp[,"bwt"], x=pdatTmp[,c("sex","age")],
+     prdat=prDat, vc=vc)
>
> par(mfrow=c(1,2))
> plot(qqInt.1/(2*log(10)),
+    gmap=gmapF8[match(rownames(qqInt.1),gmapF8$snp),],
+    main="Body Weight: Epistasis (A)\n\n",xlab="",ylab="")
> plot(qqInt.2/(2*log(10)),
+    gmap=data.frame(snp=prDat$snp,chr=prDat$chr,dist=prDat$dist),
+    main="Body Weight: Epistasis (B)\n\n",xlab="",ylab="")
```

Note that we need a genetic map 'gmap' to plot the `scanTwo` objects. Refer to the R documentation for usage of `scanTwo`.

11 Citation

Package 'QTLRel' was built on the R and C code that implemented the methodology described in Cheng et al (2010). Users of the package can kindly cite the following papers:

a) Cheng R, Abney M, Palmer AA and Skol AD (2011). QTLRel: an R Package for Genome-wide Association Studies in which Relatedness is a Concern. BMC Genet. 12: 66.

b) Cheng R, Lim JE, Samocha KE, Sokoloff G, Abney M, Skol AD and Palmer AA (2010). Genome-wide association studies and the problem of relatedness among advanced intercross lines and other highly recombinant populations. Genetics 185: 1033-1044.

We wish you success in your research!

References

Abney M, McPeek MS, Ober C (2000) Estimation of variance components of quantitative traits in inbred populations. Am J Hum Genet 141:629–650

Cheng R, Lim JE, Samocha KE, Sokoloff G, Abney M, Skol AD, Palmer AA (2010) Genome-wide association studies and the problem of relatedness among advanced intercross lines and other highly recombinant populations. Genetics 185:1033–1044

Cheng R, Abney M, Palmer AA, Skol AD (2011) Qtlrel: an R package for genome-wide association studies in which relatedness is a concern. BMC Genetics 12:66

Lynch M, Walsh B (1998) Genetics and analysis of quantitative traits, vol 5. Sinauer Associates, Inc