Preparing Case-Parent Trio Data and Detecting Disease-Associated SNPs, SNP Interactions, and Gene-Environment Interactions with `trio`

Holger Schwender, Qing Li, and Ingo Ruczinski

1 Introduction

The R package `trio` contains functions for performing genotypic transmission disequilibrium tests (gTDTs) and corresponding score tests to test whether the distributions of individual SNPs (Schaid, 1996), two-way interactions of SNPs (Cordell, 2002; Cordell *et al.*, 2004), or interactions between SNPs and binary environmental variables differ between the cases, i.e. the children affected by a disease, and the matched pseudo-controls, i.e. the combinations of alleles that were not transmitted from the parents to their offspring, but were also possible given the parents' genotypes. Additionally, `trio` also comprises a function for applying the allelic TDT (Spielman *et al.*, 1993) to genotype data.

Moreover, `trio` provides functionalities relevant for the analysis of case-parent trio data with *trio logic regression* (Li *et al.*, 2010). These are, on the one hand, functions that aid in the transformation of the trio data from standard linkage files (ped format) or genotype format into objects suitable as input for trio logic regression, and on the other hand, functions for applying trio logic regression and a bagging version of trio logic regression to these objects.

Finally, `trio` provides functions for reading in and manipulating the case-parent trio data, estimating LD as well as LD-blocks, and simulating case-parent trio data, where the risk of disease is specified by (higher order) SNP interactions.

In Section 2 of this vignette, it is shown how family-based data stored in a linkage/ped file can be read into R and transformed into a format suitable for the application of the functions for performing the allelic and genotypic TDTs as well as the score tests. While Section 3 contains examples for the application of the gTDT functions to individual SNPs, two-way SNP interactions, and gene-environment interactions, Section 4 briefly shows how the allelic TDT and the score tests related to the genotypic TDTs can be used to test these factors.

Section 5 is devoted to the steps relevant for data processing necessary to generate a matrix suitable as input for trio logic regression, starting from a linkage or genotype file, possibly containing missing data and/or Mendelian errors. We give some examples how missing data can be addressed using haplotype-based imputation. The haplotype information can be specified by the user, or when this information is not readily available, automatically inferred. The haplotype blocks are also relevant in the delineation of the genotypes for the pseudo-controls, as the linkage disequilibrium (LD) structure observed in the parents is taken into account in this process. While this function is intended to generate complete case-pseudo-control data as input for trio logic regression, an option to simply return the completed trio data is also available.

In Section 6, it is shown how trio logic regression and trioFS (trio Feature Selection; Schwender *et al.*, 2011a), in which bagging with base learner trio logic regression is used to stabilize the search for relevant SNP interactions, can be applied to the data generated as described in Section 5.

For the estimation of the haplotype structure that might be used in the functions described in Section 5, the R package `trio` also contains functions for computing and plotting the

pairwise LD values and for detecting LD blocks. In Section 7, it is described how the pairwise values of the LD measures D' and r^2 can be computed with the function getLD, and how the D' values can be employed to estimate haplotype blocks with the algorithm of Gabriel *et al.* (2002).

Finally, Section 8 of the vignette explains in more detail how to set up simulations of case-parent trio data, where the risk of disease is specified by SNP interactions. The most time-consuming step for these types of simulations is the generation of mating tables and the respective probabilities. The mating table information, however, can be stored, which allows for fast simulations when replicates of the case-parent trio data are generated.

2 Preparing Data for the Genotypic TDTs

Case-parent trio data are typically stored in a ped file. The first six columns in such a ped file, which is also referred to as linkage file, identify the family structure of the data, and the phenotype. It is assumed that only one phenotype variable (column 6) is used. The object trio.ped1, available in the R package, is an example of a data set in ped format. It contains information for 10 SNPs in 100 trios. Besides the variables providing information on the family structure and the phenotypes (columns 1–6), each SNP is encoded in two variables denoting the alleles.

```
> library(trio)
> data(trio.data)
> str(trio.ped1)

'data.frame':       300 obs. of  26 variables:
 $ famid   : int  10001 10001 10001 10002 10002 10002 10003 10003 10003 10004 ...
 $ pid     : int  1 2 3 1 2 3 1 2 3 1 ...
 $ fatid   : int  0 0 1 0 0 1 0 0 1 0 ...
 $ motid   : int  0 0 2 0 0 2 0 0 2 0 ...
 $ sex     : int  1 2 2 1 2 1 1 2 1 1 ...
 $ affected: int  0 0 2 0 0 2 0 0 2 0 ...
 $ snp1_1  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ snp1_2  : int  1 1 1 1 1 1 1 1 1 2 ...
 $ snp2_1  : int  1 2 1 1 2 1 1 1 1 1 ...
```

3

```
$ snp2_2  : int  1 2 2 1 2 2 1 1 1 1 ...
$ snp3_1  : int  1 1 1 2 1 1 1 1 1 1 ...
$ snp3_2  : int  2 1 2 2 1 2 1 2 1 2 ...
$ snp4_1  : int  1 1 1 2 1 1 1 1 1 1 ...
$ snp4_2  : int  2 1 2 2 1 2 1 2 1 2 ...
$ snp5_1  : int  1 2 1 1 2 1 1 1 1 1 ...
$ snp5_2  : int  2 2 2 1 2 2 2 1 1 1 ...
$ snp6_1  : int  1 1 1 2 1 1 1 1 1 1 ...
$ snp6_2  : int  2 1 2 2 1 2 1 2 1 2 ...
$ snp7_1  : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp7_2  : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp8_1  : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp8_2  : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp9_1  : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp9_2  : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp10_1 : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp10_2 : int  1 1 1 2 1 1 1 1 1 2 ...

> trio.ped1[1:10,1:12]
```

```
   famid pid fatid motid sex affected snp1_1 snp1_2 snp2_1 snp2_2 snp3_1 snp3_2
1  10001   1     0     0   1        0      1      1      1      1      1      2
2  10001   2     0     0   2        0      1      1      2      2      1      1
3  10001   3     1     2   2        2      1      1      1      2      1      2
4  10002   1     0     0   1        0      1      1      1      1      2      2
5  10002   2     0     0   2        0      1      1      2      2      1      1
6  10002   3     1     2   1        2      1      1      1      2      1      2
7  10003   1     0     0   1        0      1      1      1      1      1      1
8  10003   2     0     0   2        0      1      1      1      1      1      2
9  10003   3     1     2   1        2      1      1      1      1      1      1
10 10004   1     0     0   1        0      1      2      1      1      1      2
```

If not already available as data frame or matrix in the R workspace, trio data can be read into R using the function read.pedfile. If we, for example, assume that the working directory of the current R session contains a file called "pedfile.ped" (this file is actually not available in trio, we just assume that such a file exists in the working directory), then this file can be read into R by calling

```
> ped <- read.pedfile("pedfile.ped")
```

If the arguments coded and first.row of read.pedfile are not specified by the user, read.pedfile automatically tries to figures out how the alleles in the ped file are coded,

4

and whether the first row contains the SNP names (`first.row = FALSE`) or the data for the first subject (`first.row = TRUE`). In the former case, `read.pedfile` adds the SNP names (with extensions .1 and .2 to differ between the two alleles) to the respective columns of the read-in data frame.

For the applications of the functions for performing gTDTs (see Section 3), the trio data must be in a matrix in genotype format. In such a matrix, each columns represents a SNP, which is coded by the number of minor alleles, and each block of 3 consecutive rows contains the genotypes of the father, the mother, and their offspring (in this order) of one specific trio. Missing values are allowed in this matrix, and need to be coded by `NA`. This matrix can either be generated from a data frame in ped format by employing the function `ped2geno`, or more conveniently, by setting `p2g = TRUE` in `read.pedfile`. Thus, a matrix in genotype format might be obtained from the above ped file by calling

```
> geno <- read.pedfile("pedfile.ped", p2g=TRUE)
```

The output of these functions just contains the matrix in genotype format, whereas `trio.check` described in Section 5 additionally contains information about Mendelian errors. Instead of checking for Mendelian errors in `ped2geno` or `read.pedfile`, such errors are removed SNP-wise in the functions for performing genotypic TDTs.

If, for example, the data frame `trio.ped1` should be transformed into a matrix in genotype format, `ped2geno` can be applied to it. However, `ped2geno` requires unique personal IDs (second column of `trio.ped1`) such that we first have to combine the family ID and the personal ID (which would be automatically done by `read.pedfile`), and change the IDs of the fathers and mothers in columns 3 and 4 likewise.

```
> trio.ped1[,2] <- paste(trio.ped1[,1], trio.ped1[,2], sep="_")
> ids <- trio.ped1[,3] != 0
> trio.ped1[ids,3] <- paste(trio.ped1[ids,1], trio.ped1[ids,3], sep="_")
> trio.ped1[ids,4] <- paste(trio.ped1[ids,1], trio.ped1[ids,4], sep="_")
> trio.ped1[1:5, 1:4]
```

```
     famid      pid    fatid    motid
1 10001 10001_1        0        0
2 10001 10001_2        0        0
3 10001 10001_3 10001_1 10001_2
4 10002 10002_1        0        0
5 10002 10002_2        0        0
```

Afterwards, ped2geno can be applied to `trio.ped1`

```
> geno <- ped2geno(trio.ped1)
> geno[1:5,]
```

```
        SNP1 SNP2 SNP3 SNP4 SNP5 SNP6 SNP7 SNP8 SNP9 SNP10
10001_1    0    0    1    1    1    1    0    0    0     0
10001_2    0    2    0    0    2    0    0    0    0     0
10001_3    0    1    1    1    1    1    0    0    0     0
10002_1    0    0    2    2    0    2    0    0    0     1
10002_2    0    2    0    0    2    0    0    0    0     0
```

The matrix `trio.gen1` is the genotype matrix corresponding to `trio.ped1`. So the genotypes in the output of ped2geno are identical to `trio.gen1` (except for that the first two columns of `trio.gen1` contain the family ID and the personal ID).

```
> trio.gen1[1:5, 3:12]
```

```
  snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1    0    0    1    1    1    1    0    0    0     0
2    0    2    0    0    2    0    0    0    0     0
3    0    1    1    1    1    1    0    0    0     0
4    0    0    2    2    0    2    0    0    0     1
5    0    2    0    0    2    0    0    0    0     0
```

```
> table(trio.gen1[,3:12] == geno)
```

```
TRUE
3000
```

3 Testing SNPs, Pairs of SNPs, and GxE Interactions with Genotypic TDTs

A single SNP or two-way interaction can be tested with a gTDT by employing the functions `tdt` and `tdtGxG`. If we, for example, would like to test the first SNP in the matrix `mat.test` available in the R package `trio`, then this could be done by calling

```
> tdt(mat.test[,1])

        Genotypic TDT Based on 3 Pseudo Controls

Model Type: Additive

     Coef     OR  Lower Upper     SE Statistic p-Value
 -0.04256 0.9583 0.6396 1.436 0.2063   0.04255  0.8366
```

In this case, a conditional logistic regression is fitted, and the output of `tdt` contains the parameter estimate `Coef` for the SNP in this model, the odds ratio `OR`, the `Lower` and `Upper` bound of the 95% confidence interval of this odds ratio, the standard error `SE` of the parameter estimate, the Wald `Statistic` for testing whether this SNP has an effect, and the corresponding `p-Value`.

By default, an additive effect is tested. It is, however, also possible to consider a dominant effect

```
> tdt(mat.test[,1], model="dominant")

        Genotypic TDT Based on 3 Pseudo Controls

Model Type: Dominant

    Coef     OR Lower Upper     SE Statistic p-Value
 -0.1134 0.8928 0.5219 1.527 0.2739    0.1713   0.679
```

or a recessive effect

```
> tdt(mat.test[,1], model="recessive")

        Genotypic TDT Based on 3 Pseudo Controls
```

```
Model Type: Recessive

    Coef    OR  Lower Upper     SE Statistic p-Value
 0.06502 1.067 0.5279 2.157 0.3591   0.03278  0.8563
```

Similarly the interaction between SNP1 and SNP2 in `mat.test` can be tested by

```
> tdtGxG(mat.test[,1], mat.test[,2])

Genotypic TDT for Epistatic Interactions (Using 15 Pseudo Controls)

Likelihood Ratio Test:
Loglikelihood (with Interactions): -272.644
Loglikelihood (without IAs):        -275.29
Test Statistic: 5.293
P-Value:         0.26
```

In this case, the interaction is tested for epistatic interactions as described in Cordell (2002). Thus, two conditional logistic regression models are fitted to the cases and the respective 15 matched pseudo-controls (i.e. the 15 possible, but not transmitted Mendelian genotype realizations, given the parents' genotypes at the two loci), one consisting of two dummy variables for each of the two SNPs, and the other additionally containing the four possible interactions of these dummy variables. The two fitted models are then compared by a likelihood ratio test, and the p-values are computed by approximation to a χ^2-distribution with four degrees of freedom.

This is the recommended way to test the two-way interaction. `tdtGxG`, however, also provides a simpler test, in which the values of the SNPs (either coding for an additive – which is the default – for a recessive, or for a dominant model) are simply multiplied for each case and its 15 matched pseudo-controls, and a conditional logistic regression is applied to this interaction.

```
> tdtGxG(mat.test[,1], mat.test[,2], epistatic=FALSE)

Genotypic TDT for Two-Way Interaction (Using 15 Pseudo Controls)

Model Type: Additive

    Coef    OR  Lower Upper     SE Statistic p-Value
 0.02424 1.025 0.7806 1.345 0.1387   0.03052  0.8613
```

All SNPs represented by the columns of a matrix in genotype format can be tested with a gTDT by employing the function `colTDT`. Thus, all SNPs in `mat.test` can be tested by calling

```
> tdt.out <- colTDT(mat.test)
> tdt.out

        Genotypic TDT Based on 3 Pseudo Controls

Model Type: Additive

Top 5 SNPs:
          Coef     OR  Lower Upper     SE Statistic p-Value Trios
SNP6  0.44895 1.5667 0.9910 2.477 0.2337    3.6908 0.05471    63
SNP3 -0.22884 0.7955 0.5103 1.240 0.2265    1.0209 0.31232    66
SNP2 -0.19671 0.8214 0.5561 1.213 0.1990    0.9772 0.32288    73
SNP4 -0.13353 0.8750 0.5783 1.324 0.2113    0.3994 0.52740    71
SNP5  0.09764 1.1026 0.7148 1.701 0.2211    0.1950 0.65881    64
```

By default, the five top SNPs, i.e. the five SNPs with the lowest p-values, are shown ordered by their significance. The top three SNPs can be shown by

```
> print(tdt.out, 3)

        Genotypic TDT Based on 3 Pseudo Controls

Model Type: Additive

Top 3 SNPs:
         Coef     OR  Lower Upper     SE Statistic p-Value Trios
SNP6  0.4490 1.5667 0.9910 2.477 0.2337    3.6908 0.05471    63
SNP3 -0.2288 0.7955 0.5103 1.240 0.2265    1.0209 0.31232    66
SNP2 -0.1967 0.8214 0.5561 1.213 0.1990    0.9772 0.32288    73
```

If the integer specified in `print` is larger than or equal to the number of SNPs in the input matrix, the statistics for all SNPs are displayed in the order of their appearance in this matrix.

```
> print(tdt.out, 10)

        Genotypic TDT Based on 3 Pseudo Controls

Model Type: Additive

            Coef     OR  Lower Upper     SE Statistic p-Value Trios
```

```
SNP1 -0.04256 0.9583 0.6396 1.436 0.2063   0.04255 0.83658   72
SNP2 -0.19671 0.8214 0.5561 1.213 0.1990   0.97724 0.32288   73
SNP3 -0.22884 0.7955 0.5103 1.240 0.2265   1.02085 0.31232   66
SNP4 -0.13353 0.8750 0.5783 1.324 0.2113   0.39941 0.52740   71
SNP5  0.09764 1.1026 0.7148 1.701 0.2211   0.19497 0.65881   64
SNP6  0.44895 1.5667 0.9910 2.477 0.2337   3.69084 0.05471   63
```

Since the genetic mode of inheritance is typically unknown, it might be beneficial to use the maximum over the gTDT statistics for an additive, a dominant, and a recessive effect as test statistic, which can be done using the function `colTDTmaxStat`

```
> max.stat <- colTDTmaxStat(mat.test)
> max.stat

          Maximum Genotypic TDT Statistic

Top 5 SNPs:
     Max-Statistic Additive Dominant Recessive
SNP6        5.1295   3.6908  1.14571   5.12953
SNP2        3.1569   0.9772  0.04811   3.15688
SNP3        2.7150   1.0209  2.71503   0.76851
SNP4        0.6990   0.3994  0.69897   0.01234
SNP5        0.2156   0.1950  0.07337   0.21555
```

This function just computes the MAX gTDT statistic, i.e. the maximum over the three gTDT statistics, since in contrast to these gTDT statistics, which under the null hypothesis follow an asymptotic χ_1^2-distribution, the null distribution of the MAX gTDT statistic is unknown, and must therefore be estimated by a (time-consuming) permutation procedure. To also determine permutation-based p-values, `colTDTmaxTest` can be applied to a matrix in genotype matrix. For example,

```
> max.out <- colTDTmaxTest(mat.test, perm=1000)
```

computes p-values for the six SNPs in `mat.test` based on 1000 permutations of the case-pseudo-control status.

```
> max.out

          Maximum Genotypic TDT

Top 5 SNPs:
```

```
        Max-Statistic Additive Dominant Recessive p-Value
SNP6          5.1295   3.6908  1.14571   5.12953   0.059
SNP2          3.1569   0.9772  0.04811   3.15688   0.146
SNP3          2.7150   1.0209  2.71503   0.76851   0.199
SNP4          0.6990   0.3994  0.69897   0.01234   0.696
SNP5          0.2156   0.1950  0.07337   0.21555   0.878
```

All two-way interactions comprised a matrix in genotype format can be tested using the function colGxG. Since both the gTDT for two-way interactions and the likelihood ratio test of Cordell *et al.* (2004) assume that the two considered loci are unlinked, the testing might fail, i.e. the fitting of the conditional logistic regression might not work properly, if the two SNPs are in (strong) LD. (Another reason why the fitting might not work properly is that the minor allele frequencies of both SNPs are very small.) Therefore, colGxG provides an argument called genes that allows specifying which SNP belongs to which LD-block, gene, or genetic region. If genes is not specified, the interactions between all $m(m-1)/2$ pairs of the m SNPs in a matrix are tested. If specified, only the interactions between SNPs showing different values of genes are tested.

If we thus assume that the first two SNPs in mat.test belong to gene G1 and the other four SNPs to G2

```
> genes <- paste("G", rep(1:2, c(2,4)), sep="")
> genes

[1] "G1" "G1" "G2" "G2" "G2" "G2"
```

then only the four interactions between SNP1 and each SNP from gene G2, as well as the four interactions between SNP2 and each SNP from gene G2 are tested, when calling

```
> tdt2.out <- colGxG(mat.test, genes=genes)
> tdt2.out

        Genotypic TDT for Epistatic Interactions (Using 15 Pseudo Controls)

Top 5 SNP Interactions (Likelihood Ratio Test):
            LL (with IAs) LL (w/o IAs) Statistic  P-Value   Genes
SNP1 : SNP5        -269.5       -277.0    15.069 0.004561 G1 : G2
SNP2 : SNP4        -270.3       -275.0     9.528 0.049167 G1 : G2
SNP1 : SNP3        -273.0       -275.2     4.440 0.349724 G1 : G2
```

```
SNP2 : SNP5        -273.3        -275.3      3.871 0.423763 G1 : G2
SNP2 : SNP6        -271.2        -272.6      2.805 0.591008 G1 : G2
```

Again, by default the top five SNP interactions are shown. The statistics for all eight interactions can be displayed by calling

```
> print(tdt2.out, 8)

        Genotypic TDT for Epistatic Interactions (Using 15 Pseudo Controls)

Likelihood Ratio Test:
            LL (with IAs) LL (w/o IAs) Statistic  P-Value   Genes
SNP1 : SNP3        -273.0        -275.2      4.440 0.349724 G1 : G2
SNP1 : SNP4        -275.9        -276.8      1.653 0.799239 G1 : G2
SNP1 : SNP5        -269.5        -277.0     15.069 0.004561 G1 : G2
SNP1 : SNP6        -273.3        -274.3      2.050 0.726494 G1 : G2
SNP2 : SNP3        -273.0        -273.4      0.778 0.941371 G1 : G2
SNP2 : SNP4        -270.3        -275.0      9.528 0.049167 G1 : G2
SNP2 : SNP5        -273.3        -275.3      3.871 0.423763 G1 : G2
SNP2 : SNP6        -271.2        -272.6      2.805 0.591008 G1 : G2
```

In genetic association studies, it is often also of interest to test gene-environment interactions, where most of the usually considered environmental variables are binary. The R package trio therefore also provides a function called colGxE to test the interactions between each of the SNPs comprised by a matrix in genotype format and a binary environmental variable with values zero and one. If we, for example, assume that the children in the first 50 trios comprised by (the first 150 rows of) mat.test are girls, and the remaining 50 are boys,

```
> sex <- rep(0:1, e=50)
```

then we can test the interactions between the six SNPs in mat.test and the environmental variable "sex" by

```
> gxe.out <- colGxE(mat.test, sex)
> gxe.out

        Genotypic TDT for GxE Interactions with Binary E

Model Type: Additive

Top 5 GxE Interactions:
```

```
            Coef     OR  Lower Upper     SE Statistic p-value Trios0 Trios1
SNP2   0.5849 1.7949 0.8134 3.961 0.4038    2.0982  0.1475     37     36
SNP1  -0.4257 0.6533 0.2896 1.474 0.4151    1.0518  0.3051     32     40
SNP6  -0.3878 0.6786 0.2697 1.708 0.4708    0.6783  0.4102     34     29
SNP4   0.2624 1.3000 0.5668 2.982 0.4235    0.3838  0.5356     36     35
SNP5   0.2007 1.2222 0.5129 2.912 0.4430    0.2052  0.6506     33     31


Effects of the SNPs in the Corresponding GxE Models:
            Coef     OR  Lower Upper     SE Statistic p-value
SNP2  -0.5108 0.6000 0.3345 1.076 0.2981    2.9356 0.08665
SNP1   0.1744 1.1905 0.6664 2.127 0.2960    0.3469 0.55585
SNP6   0.6242 1.8667 0.9970 3.495 0.3200    3.8051 0.05110
SNP4  -0.2624 0.7692 0.4294 1.378 0.2974    0.7781 0.37771
SNP5   0.0000 1.0000 0.5462 1.831 0.3086    0.0000 1.00000


ORs for Exposed Cases:
           OR  Lower Upper
SNP2 1.0769 0.6315 1.837
SNP1 0.7778 0.4397 1.376
SNP6 1.2667 0.6437 2.493
SNP4 1.0000 0.5538 1.806
SNP5 1.2222 0.6556 2.279


2 df Likelihood Ratio Test, 2 df Wald Test, 1 df Likelihood Ratio Test:
     2df Stat 2df p-Value Wald Stat Wald p-value 1df Stat 1df p-Value
SNP2   3.1061      0.2116    3.0096       0.2221   2.1242      0.1450
SNP1   1.1002      0.5769    1.0930       0.5790   1.0577      0.3037
SNP6   4.4641      0.1073    4.2735       0.1180   0.6797      0.4097
SNP4   0.7848      0.6754    0.7781       0.6777   0.3845      0.5352
SNP5   0.4007      0.8185    0.3987       0.8193   0.2055      0.6503
```

In this situation, a conditional logistic regression model $\beta_{\mathsf{G}} G + \beta_{\mathsf{GxE}}(G \times E)$ is fitted for each SNP, where G is a variable coding for an additive effect of the SNP, and $G \times E$ is the corresponding gene-environment interaction. Analogously to the other gTDT functions, a dominant or a recessive effect can also be considered by changing the argument `model` of `colGxE`.

For both β_{G} and β_{GxE}, the same as stastistics as, for example, in `colTDT`, are ocmputed. Additionally, the odds ratios and their confidence intervals for the exposed trios are determined (note that the odds ratios for the unexposed trios are given by $\exp(\beta_{\mathsf{G}})$), and a 2 degree of freedom Wald test for testing both β_{G} and β_{GxE} simultaneously as well as

two likelihood ratio tests are performed, where the 2 df likelihood ratio test compares the likelihood of the full model (containing both β_G and β_{GxE}) with the likelihood of a null model containing no variable, and the 1 df likelihood ratio test compares the likelihoods of the full model and a model only consisting of the SNP. The computation of all these statistics can be avoided by setting addGandE = FALSE (for the odds ratio of the exposed cases), add2df = FALSE (for the 2 df Wald test) and whichLRT = "none" for the two likelihood ratio tests.

If these statistics should be determined, but only the results for the genotypic TDT for testing the gene-environment interaction should be printed, then this can be done by calling

```
> print(gxe.out, onlyGxE=TRUE)

          Genotypic TDT for GxE Interactions with Binary E

Model Type: Additive

Top 5 GxE Interactions:
        Coef     OR  Lower Upper     SE Statistic p-value Trios0 Trios1
SNP2  0.5849 1.7949 0.8134 3.961 0.4038    2.0982  0.1475     37     36
SNP1 -0.4257 0.6533 0.2896 1.474 0.4151    1.0518  0.3051     32     40
SNP6 -0.3878 0.6786 0.2697 1.708 0.4708    0.6783  0.4102     34     29
SNP4  0.2624 1.3000 0.5668 2.982 0.4235    0.3838  0.5356     36     35
SNP5  0.2007 1.2222 0.5129 2.912 0.4430    0.2052  0.6506     33     31
```

A convenient way to generate a data frame containing all the statistics determined by colGxE is to use the function getGxEstats. This function can be employed to obtain these results for all considered SNPs, and it can also be used to get these statistics only for the top SNPs. If, for example, the results of the three SNPs showing the smallest p-values for the 2 df likelihood ratio test are of interest, then the data frame containing these SNPs can be generated by

```
> dat.top3 <- getGxEstats(gxe.out, top=3, sortBy="lrt2df")
> dat.top3

         Coef GxE    OR GxE Lower GxE Upper GxE    SE GxE  Stat GxE  pval GxE
SNP6 -0.3877655 0.6785714 0.2696658  1.707518 0.4708282 0.6782869 0.4101772
```

```
SNP2   0.5849336 1.7948718 0.8134061   3.960586 0.4038127 2.0982302 0.1474697
SNP1  -0.4256678 0.6533333 0.2896241   1.473787 0.4150604 1.0517658 0.3051008
      Trios0 Trios1      Coef G      OR G   Lower G  Upper G      SE G    Stat G
SNP6      34     29  0.6241543 1.866667 0.9970286 3.494829 0.3199702 3.8050887
SNP2      37     36 -0.5108256 0.600000 0.3344814 1.076293 0.2981424 2.9356067
SNP1      32     40  0.1743534 1.190476 0.6664394 2.126575 0.2960051 0.3469463
         pval G    OR G&E Lower G&E Upper G&E Stat LRT 2df pval LRT 2df
SNP6 0.05109707 1.2666667 0.6436665  2.492664     4.464086     0.1073090
SNP2 0.08664610 1.0769231 0.6314775  1.836587     3.106149     0.2115964
SNP1 0.55584687 0.7777778 0.4397363  1.375684     1.100231     0.5768831
     Stat Wald 2df pval Wald 2df Stat LRT 1df pval LRT 1df
SNP6      4.273492     0.1180383    0.6797377    0.4096771
SNP2      3.009647     0.2220565    2.1241807    0.1449903
SNP1      1.093011     0.5789694    1.0576748    0.3037461
```

4   Score Tests and the Allelic TDT

Besides functions for performing genotypic TDTs, `trio` also provides functions for applying an allelic TDT and score tests related to the genotypic TDTs to matrices in genotype format. For example, the case-parent trio data in `mat.test` can be analyzed with an allelic TDT by

```
> a.out <- allelicTDT(mat.test)
> a.out

        Allelic TDT

Top 5 SNPs:
     Statistic p-value
SNP6    3.3247 0.06825
SNP3    0.8101 0.36808
SNP2    0.7941 0.37286
SNP4    0.2778 0.59816
SNP5    0.1098 0.74042
```

By default, the test statistic for the allelic TDT is adjusted for continuity by adding 1 to the difference in the denominator of McNemar's test statistic, i.e.

$$a_{\mathsf{Cor}} = \frac{(|b - c| - 1)^2}{b + c}$$

is used as test statstic, where b and c are the off-diagonal elements of the 2x2-table summarizing the transmitted and untransmitted alleles from heterozygous parents, i.e. b is the number of heterozygous parents that transmitted the minor allele to their respective children, and c the number of heterozygous parents that transmitted the major allele. This correction can be avoided and

$$a = \frac{(b-c)^2}{b+c}$$

can be used as test statistic by calling

```
> a.out2 <- allelicTDT(mat.test, correct=FALSE)
> a.out2

        Allelic TDT

Top 5 SNPs:
     Statistic p-value
SNP6    3.7532 0.05271
SNP3    1.0253 0.31126
SNP2    0.9804 0.32210
SNP4    0.4000 0.52709
SNP5    0.1951 0.65869
```

If a score test instead of a Wald test, i.e. a genotypic TDT, should be considered in the analysis of genotype data, then scoreTDT, scoreMaxStat, scoreGxG, and scoreGxE can be used instead of and in the same way as colTDT, colTDTmaxStat, colGxG, and colGxE, respectively. For example, the SNPs in mat.test can be tested individually under the assumption of an additive mode of inheritance by

```
> s.out <- scoreTDT(mat.test)
> s.out

    Score Test for Individual SNPs

Model Type: Additive

Top 5 SNPs:
     Score Statistic p-value
SNP6   8.5    3.7532 0.05271
SNP3  -4.5    1.0253 0.31126
SNP2  -5.0    0.9804 0.32210
SNP4  -3.0    0.4000 0.52709
SNP5   2.0    0.1951 0.65869
```

Score tests, however, only provide scores and p-values, but do not the allow the computation of odds ratios, relative risks, and confidence intervals. Usually, score tests have the advantage that their test statistic is much faster too compute than the statistic of a Wald test. However, when considering SNPs or interactions between SNPs and binary environmental factors, both the score tests and the genotypic TDTs have about the same computation time, as in these situations, there also exist analytic solutiongs for the genotypic TDT statistics (see Schwender *et al.*, 2011b).

5 Generating Data for Trio Logic Regression Input

If interactions of a higher order than two are of interest, trio logic regression can be used to detect disease-associated SNP interactions of any order.

To generate data that can be used as input in trio logic regression, the sequential application of two functions is required. The function `trio.check` evaluates whether or not Mendelian errors are present in the data (stored either in linkage or in genotype format, see Section 5.1). If no Mendelian inconsistencies are detected, this function creates an object that is passed to the function `trio.prepare`. The latter function then generates a matrix of the genotype information for the affected probands and the inferred pseudo-controls, taking the observed LD structure into account. Missing data are imputed in the process. The user, however, has to supply the information for the lengths of the LD blocks. A function called `findLDblocks` for identifying LD blocks, and thus, for specifying the length of the blocks is therefore also contained in this package (see Section 7). Given the lengths of the LD blocks, the haplotype frequencies can be estimated, using the function `haplo.em` in the `haplo.stat` package.

5.1 Supported File Formats and Elementary Data Processing

In this section, we show how to generate data suitable for input to trio logic regression from complete pedigree data without Mendelian errors. The function `trio.check` requires that the trio data are already available as a data frame or matrix, either in linkage/ped format (the default), or in genotype format (for reading a ped file into R, see Section 3).

The first function used is always `trio.check`. Unless otherwise specified, this function assumes that the data are in linkage format. If no Mendelian inconsistencies in the data provided are identified, `trio.check` creates an object that can be processed in the subsequent analysis with this package. The genotype information for each SNP will be converted into a single variable, denoting the number of variant alleles.

If we thus would like to check whether the data frame `trio.ped1` contains Mendelian errors, we call

```
> trio.tmp <- trio.check(dat=trio.ped1)
> str(trio.tmp, max=1)

List of 2
 $ trio  :'data.frame':        300 obs. of  12 variables:
 $ errors: NULL

> trio.tmp$trio[1:6,]

   famid      pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1 10001 10001_1    0    0    1    1    1    1    0    0    0     0
2 10001 10001_2    0    2    0    0    2    0    0    0    0     0
3 10001 10001_3    0    1    1    1    1    1    0    0    0     0
4 10002 10002_1    0    0    2    2    0    2    0    0    0     1
5 10002 10002_2    0    2    0    0    2    0    0    0    0     0
6 10002 10002_3    0    1    1    1    1    1    0    0    0     0
```

Taking the LD structure of the SNPs into account is imperative when creating the genotypes for the pseudo-controls. This requires information on the LD blocks. However, there are many ways to delineate this block structure, and in the absence of a consensus what the best approach is, researchers have different preferences, and thus, results can be

different. In the function `findLDblocks`, a modified version of the method of Gabriel *et al.* (2002) has been implemented, which can be used to specify the block structure by

```
> table(foundBlocks$blocks}
```

if `foundBlocks` is the output of `findLDblocks` (for details, see Section 7).

The function `trio.prepare`, which operates on an output object of trio.check, accepts the block length information as an argument (in the following, we assume that the block structure is given by `c(1, 4, 2, 3)`, i.e. the first block consists only of the first SNP, the second block of the next four SNPs, the third of the following two SNPs, and the last block of the remaining three SNPs). If this argument is not specified, a uniform block length of 1 (i.e. no LD structure) is assumed. If the haplotype frequencies are not specified, they are estimated from the parents' genotypes (more information on this in the following sections). The function `trio.prepare` then returns a list that contains the genotype information in binary format, suitable as input for trio logic regression: `bin` is a matrix with the conditional logistic regression response in the first columns, and each SNP as two binary variables using dominant and recessive coding. The list element `miss` contains information about missing values in the original data, and `freq` contains information on the estimated haplotype frequencies. To make the matrix generated by `trio.prepare` reproducible, the function `set.seed` is used to set the random number generator in a reproducible state.

```
> set.seed(123456)
> trio.bin <- trio.prepare(trio.dat=trio.tmp, blocks=c(1,4,2,3))
> str(trio.bin, max=1)

List of 3
 $ bin : num [1:400, 1:21] 3 0 0 0 3 0 0 0 3 0 ...
  ..- attr(*, "dimnames")=List of 2
 $ miss: NULL
 $ freq:'data.frame':        19 obs. of  3 variables:
 - attr(*, "class")= chr "trioPrepare"

> trio.bin$bin[1:8,]
```

```
       y snp1.D snp1.R snp2.D snp2.R snp3.D snp3.R snp4.D snp4.R snp5.D snp5.R
[1,] 3       0       0       1       0       1       0       1       0       1       0
[2,] 0       0       0       1       0       0       0       0       0       1       1
[3,] 0       0       0       1       0       1       0       1       0       1       0
[4,] 0       0       0       1       0       0       0       0       0       1       1
[5,] 3       0       0       1       0       1       0       1       0       1       0
[6,] 0       0       0       1       0       1       0       1       0       1       0
[7,] 0       0       0       1       0       1       0       1       0       1       0
[8,] 0       0       0       1       0       1       0       1       0       1       0
     snp6.D snp6.R snp7.D snp7.R snp8.D snp8.R snp9.D snp9.R snp10.D snp10.R
[1,]      1       0       0       0       0       0       0       0        0        0
[2,]      0       0       0       0       0       0       0       0        0        0
[3,]      0       0       0       0       0       0       0       0        0        0
[4,]      1       0       0       0       0       0       0       0        0        0
[5,]      1       0       0       0       0       0       0       0        0        0
[6,]      1       0       0       0       0       0       0       0        0        0
[7,]      1       0       0       0       0       0       0       0        1        0
[8,]      1       0       0       0       0       0       0       0        1        0
```

As mentioned above, the `trio` package also accommodates trio genotype data. The object `trio.gen1`, available in the R package, is an example of such a data set. Equivalent to `trio.ped1` used above, it contains information for 10 SNPs in 100 trios. When used in `trio.check`, the argument `is.linkage` needs to be set to `FALSE`. The output from this function is then identical to the one shown derived from the linkage file, and can be passed to the function `trio.prepare`.

```
> str(trio.gen1)

'data.frame':        300 obs. of  12 variables:
 $ famid: int  10001 10001 10001 10002 10002 10002 10003 10003 10003 10004 ...
 $ pid  : int  1 2 3 1 2 3 1 2 3 1 ...
 $ snp1 : int  0 0 0 0 0 0 0 0 0 1 ...
 $ snp2 : int  0 2 1 0 2 1 0 0 0 0 ...
 $ snp3 : int  1 0 1 2 0 1 0 1 0 1 ...
 $ snp4 : int  1 0 1 2 0 1 0 1 0 1 ...
 $ snp5 : int  1 2 1 0 2 1 1 0 0 0 ...
 $ snp6 : int  1 0 1 2 0 1 0 1 0 1 ...
 $ snp7 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ snp8 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ snp9 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ snp10: int  0 0 0 1 0 0 0 0 0 1 ...

> trio.gen1[1:10,1:12]

   famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1  10001   1    0    0    1    1    1    1    0    0    0     0
```

```
 2  10001   2   0   2   0   0   2   0   0   0   0   0
 3  10001   3   0   1   1   1   1   1   0   0   0   0
 4  10002   1   0   0   2   2   0   2   0   0   0   1
 5  10002   2   0   2   0   0   2   0   0   0   0   0
 6  10002   3   0   1   1   1   1   1   0   0   0   0
 7  10003   1   0   0   0   0   1   0   0   0   0   0
 8  10003   2   0   0   1   1   0   1   0   0   0   0
 9  10003   3   0   0   0   0   0   0   0   0   0   0
10  10004   1   1   0   1   1   0   1   0   0   0   1

> trio.tmp <- trio.check(dat=trio.gen1, is.linkage=FALSE)
> set.seed(123456)
> trio.bin2 <- trio.prepare(trio.dat=trio.tmp, blocks=c(1,4,2,3))
> trio.bin2$bin[1:8,]

     y snp1.D snp1.R snp2.D snp2.R snp3.D snp3.R snp4.D snp4.R snp5.D snp5.R
[1,] 3      0      0      1      0      1      0      1      0      1      0
[2,] 0      0      0      1      0      0      0      0      0      1      1
[3,] 0      0      0      1      0      1      0      1      0      1      0
[4,] 0      0      0      1      0      0      0      0      0      1      1
[5,] 3      0      0      1      0      1      0      1      0      1      0
[6,] 0      0      0      1      0      1      0      1      0      1      0
[7,] 0      0      0      1      0      1      0      1      0      1      0
[8,] 0      0      0      1      0      1      0      1      0      1      0
     snp6.D snp6.R snp7.D snp7.R snp8.D snp8.R snp9.D snp9.R snp10.D snp10.R
[1,]      1      0      0      0      0      0      0      0       0       0
[2,]      0      0      0      0      0      0      0      0       0       0
[3,]      0      0      0      0      0      0      0      0       0       0
[4,]      1      0      0      0      0      0      0      0       0       0
[5,]      1      0      0      0      0      0      0      0       0       0
[6,]      1      0      0      0      0      0      0      0       0       0
[7,]      1      0      0      0      0      0      0      0       1       0
[8,]      1      0      0      0      0      0      0      0       1       0
```

5.2   Missing Genotype Information

Missing genotypes in ped(igree) files are typically encoded using the integer 0. The data
files can be processed as before if they contain such missing values:

```
> str(trio.ped2)

'data.frame':        300 obs. of  26 variables:
 $ famid   : int  10001 10001 10001 10002 10002 10002 10003 10003 10003 10004 ...
 $ pid     : int  1 2 3 1 2 3 1 2 3 1 ...
 $ fatid   : int  0 0 1 0 0 1 0 0 1 0 ...
 $ motid   : int  0 0 2 0 0 2 0 0 2 0 ...
```

```
$ sex     : int  1 2 2 1 2 1 1 2 1 1 ...
$ affected: int  0 0 2 0 0 2 0 0 2 0 ...
$ snp1_1  : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp1_2  : int  1 1 1 1 1 1 1 1 1 2 ...
$ snp2_1  : int  1 0 1 1 2 1 1 1 1 1 ...
$ snp2_2  : int  1 0 2 1 2 2 1 1 1 1 ...
$ snp3_1  : int  1 1 1 2 0 1 1 1 1 1 ...
$ snp3_2  : int  2 1 2 2 0 2 1 2 1 2 ...
$ snp4_1  : int  1 0 1 2 1 1 1 1 1 1 ...
$ snp4_2  : int  2 0 2 2 1 2 1 2 1 2 ...
$ snp5_1  : int  1 2 1 1 2 1 1 1 1 1 ...
$ snp5_2  : int  2 2 2 1 2 2 2 1 1 1 ...
$ snp6_1  : int  1 1 1 0 1 1 1 1 1 1 ...
$ snp6_2  : int  2 1 2 0 1 2 1 2 1 2 ...
$ snp7_1  : int  1 1 1 1 1 1 1 1 0 1 ...
$ snp7_2  : int  1 1 1 1 1 1 1 1 0 1 ...
$ snp8_1  : int  1 1 1 1 0 1 1 1 0 1 ...
$ snp8_2  : int  1 1 1 1 0 1 1 1 0 1 ...
$ snp9_1  : int  1 1 1 1 1 1 1 0 1 1 ...
$ snp9_2  : int  1 1 1 1 1 1 1 0 1 1 ...
$ snp10_1 : int  1 1 1 1 1 1 1 1 1 1 ...
$ snp10_2 : int  1 1 1 2 1 1 1 1 1 2 ...

> trio.tmp <- trio.check(dat=trio.ped2)
> trio.tmp$trio[1:6,]

  famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1 10001   1    0    0    1    1    1    1    0    0    0     0
2 10001   2    0   NA    0   NA    2    0    0    0    0     0
3 10001   3    0    1    1    1    1    1    0    0    0     0
4 10002   1    0    0    2    2    0   NA    0    0    0     1
5 10002   2    0    2   NA    0    2    0    0   NA    0     0
6 10002   3    0    1    1    1    1    1    0    0    0     0
```

Since trio logic regression requires complete data, the function `trio.prepare` also performs an imputation of the missing genotypes. The imputation is based on estimated haplotypes, using the block length information specified by the user. In a later section we demonstrate how this imputation can be run more efficiently when haplotype frequency estimates are already available.

```
> set.seed(123456)
> trio.bin3 <- trio.prepare(trio.dat=trio.tmp, blocks=c(1,4,2,3))
> trio.bin3$bin[1:8,]

      y snp1.D snp1.R snp2.D snp2.R snp3.D snp3.R snp4.D snp4.R snp5.D snp5.R
[1,] 3      0      0      1      0      1      0      1      0      1      0
```

```
[2,] 0      0      0      1      0      0      0      0      0      1      1
[3,] 0      0      0      0      0      1      0      1      0      1      0
[4,] 0      0      0      0      0      0      0      0      0      1      1
[5,] 3      0      0      1      0      1      0      1      0      1      0
[6,] 0      0      0      1      0      1      0      1      0      1      0
[7,] 0      0      0      1      0      1      0      1      0      1      0
[8,] 0      0      0      1      0      1      0      1      0      1      0
     snp6.D snp6.R snp7.D snp7.R snp8.D snp8.R snp9.D snp9.R snp10.D snp10.R
[1,]      1      0      0      0      0      0      0      0       0       0
[2,]      1      0      0      0      0      0      0      0       0       0
[3,]      0      0      0      0      0      0      0      0       0       0
[4,]      0      0      0      0      0      0      0      0       0       0
[5,]      1      0      0      0      0      0      0      0       0       0
[6,]      1      0      0      0      0      0      0      0       1       0
[7,]      1      0      0      0      0      0      0      0       0       0
[8,]      1      0      0      0      0      0      0      0       1       0
```

Missing data in genotypes files should be encoded using NA, the conventional symbol in R to indicate missing values.

```
> str(trio.gen2)

'data.frame':        300 obs. of  12 variables:
 $ famid: int  10001 10001 10001 10002 10002 10002 10003 10003 10003 10004 ...
 $ pid  : int  1 2 3 1 2 3 1 2 3 1 ...
 $ snp1 : int  0 0 0 0 0 0 0 0 0 1 ...
 $ snp2 : int  0 2 1 NA NA 1 0 0 0 0 ...
 $ snp3 : int  1 NA 1 2 0 1 0 NA 0 1 ...
 $ snp4 : int  1 0 1 NA 0 1 0 1 0 1 ...
 $ snp5 : int  1 2 1 0 2 1 1 NA 0 0 ...
 $ snp6 : int  1 0 1 NA 0 1 0 1 0 1 ...
 $ snp7 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ snp8 : int  0 0 NA 0 0 0 0 NA 0 0 ...
 $ snp9 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ snp10: int  0 0 0 1 0 0 0 0 0 1 ...

> trio.tmp <- trio.check(dat=trio.gen2, is.linkage=FALSE)
> set.seed(123456)
> trio.bin4 <- trio.prepare(trio.dat=trio.tmp, blocks=c(1,4,2,3))
> trio.bin4$bin[1:8,]

     y snp1.D snp1.R snp2.D snp2.R snp3.D snp3.R snp4.D snp4.R snp5.D snp5.R
[1,] 3      0      0      1      0      1      0      1      0      1      0
[2,] 0      0      0      1      0      0      0      0      0      1      1
[3,] 0      0      0      1      0      0      0      0      0      1      1
[4,] 0      0      0      1      0      1      0      1      0      1      0
[5,] 3      0      0      1      0      1      0      1      0      1      0
[6,] 0      0      0      1      0      1      0      1      0      1      0
[7,] 0      0      0      1      0      1      0      1      0      1      0
```

```
[8,] 0       0       0       1       0       1       0       1       0       1       0
    snp6.D snp6.R snp7.D snp7.R snp8.D snp8.R snp9.D snp9.R snp10.D snp10.R
[1,]      1      0      0      0      0      0      0      0       0       0
[2,]      0      0      0      0      0      0      0      0       0       0
[3,]      0      0      0      0      0      0      0      0       0       0
[4,]      1      0      0      0      0      0      0      0       0       0
[5,]      1      0      0      0      0      0      0      0       0       0
[6,]      1      0      0      0      0      0      0      0       0       0
[7,]      0      0      0      0      0      0      0      0       1       0
[8,]      0      0      0      0      0      0      0      0       1       0
```

As the user might also be interested in the completed genotype data in the original format (genotype or linkage file), the function `trio.prepare` also allows for this option by using the argument `logic = FALSE`. In the resulting object, the matrix `bin` is then replaced by the data frame `trio`, and `miss` and `freq` are also returned.

```
> trio.tmp <- trio.check(dat=trio.gen2, is.linkage=FALSE)
> set.seed(123456)
> trio.imp <- trio.prepare(trio.dat=trio.tmp, blocks=c(1,4,2,3), logic=FALSE)
> str(trio.imp, max=1)

List of 3
 $ trio:'data.frame':        300 obs. of  12 variables:
 $ miss:'data.frame':        250 obs. of  5 variables:
 $ freq:'data.frame':         19 obs. of  3 variables:
 - attr(*, "class")= chr "trioPrepare"

> trio.imp$miss[c(1:6),]

  famid pid snp r   c
1 10001   2   3 2   5
2 10001   3   8 3  10
3 10002   1   2 4   4
4 10002   1   4 4   6
5 10002   1   6 4   8
6 10002   2   2 5   4

> trio.gen2[1:6,]

  famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1 10001   1    1    0    0    1    1    1    1    0    0     0
2 10001   2    0    2   NA    0    2    0    0    0    0     0
3 10001   3    0    1    1    1    1    1    0   NA    0     0
4 10002   1    0   NA    2   NA    0   NA    0    0    0     1
5 10002   2    0   NA    0    0    2    0    0    0    0     0
6 10002   3    0    1    1    1    1    1    0    0    0     0
```

```
> trio.imp$trio[1:6,]

  famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1 10001   1    0    0    1    1    1    1    0    0    0     0
2 10001   2    0    2    0    0    2    0    0    0    0     0
3 10001   3    0    1    1    1    1    1    0    0    0     0
4 10002   1    0    0    2    2    0    1    0    0    0     1
5 10002   2    0    1    0    0    2    0    0    0    0     0
6 10002   3    0    1    1    1    1    1    0    0    0     0
```

The same applies to pedigree data:

```
> trio.tmp <- trio.check(dat=trio.ped2)
> set.seed(123456)
> trio.imp2 <- trio.prepare(trio.dat=trio.tmp, blocks=c(1,4,2,3), logic=FALSE)
> trio.imp$trio[1:6,]

  famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1 10001   1    0    0    1    1    1    1    0    0    0     0
2 10001   2    0    2    0    0    2    0    0    0    0     0
3 10001   3    0    1    1    1    1    1    0    0    0     0
4 10002   1    0    0    2    2    0    1    0    0    0     1
5 10002   2    0    1    0    0    2    0    0    0    0     0
6 10002   3    0    1    1    1    1    1    0    0    0     0
```

5.3   Mendelian Errors

To delineate the genotype information for the pseudo-controls, the trio data must not
contain any Mendelian errors. The function trio.check returns a warning, and an R
object with relevant information when Mendelian errors are encountered is created.

```
> trio.tmp <- trio.check(dat=trio.ped.err)

[1] "Found Mendelian error(s)."

> str(trio.tmp, max=1)

List of 3
 $ trio    : NULL
 $ errors  :'data.frame':      4 obs. of  5 variables:
 $ trio.err:'data.frame':    300 obs. of  12 variables:

> trio.tmp$errors
```

```
     trio famid snp r   c
1      1 10001    9 1 11
2      1 10001   10 1 12
3      2 10002   10 4 12
4      3 10003   10 7 12
```

In this data set, trio 1, for example, contains two Mendelian errors, in SNPs 9 and 10.

```
> trio.tmp$trio.err[1:3, c(1,2, 11:12)]

   famid pid snp9 snp10
1 10001   1    0     1
2 10001   2    0     2
3 10001   3    2     0

> trio.ped.err[1:3,c(1:2, 23:26)]

   famid pid snp9_1 snp9_2 snp10_1 snp10_2
1 10001   1      1      1       1       2
2 10001   2      1      1       2       2
3 10001   3      2      2       1       1
```

It is the user's responsibility to find the cause for the Mendelian errors and correct those, if possible. However, Mendelian inconsistencies are often due to genotyping errors and thus, it might not be possible to correct those in a very straight-forward manner. In this instance, the user might want to encode the genotypes that cause theses Mendelian errors in some of the trios as missing data. The argument replace = TRUE in trio.check allows for this possibility. The resulting missing data can then be imputed as described in the previous section.

```
> trio.rep <- trio.check(dat=trio.ped.err, replace=TRUE)
> str(trio.rep, max=1)

List of 2
 $ trio  :'data.frame':        300 obs. of  12 variables:
 $ errors: NULL

> trio.rep$trio[1:3,11:12]

   snp9 snp10
1    NA    NA
2    NA    NA
3    NA    NA
```

The same option is available for data in genotype format with Mendelian inconsistencies.

```
> trio.tmp <- trio.check(dat=trio.gen.err, is.linkage=FALSE)

[1] "Found Mendelian error(s)."

> trio.tmp$errors

  trio famid snp r c
1    1  2001  5 1 7
2    2  2002  5 4 7

> trio.tmp$trio.err[1:6, c(1,2,7), drop=F]

   famid pid snp5
6   2001   1    0
7   2001   2    0
5   2001   3    1
9   2002   1    1
10  2002   2    0
8   2002   3    2

> trio.rep <- trio.check(dat=trio.gen.err, is.linkage=FALSE, replace=TRUE)
> trio.rep$trio[1:6,c(1,2,7)]

   famid pid snp5
6   2001   1   NA
7   2001   2   NA
5   2001   3   NA
9   2002   1   NA
10  2002   2   NA
8   2002   3   NA
```

5.4   Using Haplotype Frequencies

As mentioned above, when estimates for the haplotype frequencies are already available, they can be used in the imputation of missing data and the delineation of the pseudo-controls. In case there are blocks of length one, i.e. SNPs not belonging to any LD blocks, the minor allele frequencies of those SNPs are supplied. In this case, no haplotype estimation is required when the function trio.prepare is run, which can result in substantial time savings.

As an example for the format of a file containing haplotype frequency estimates and SNP minor allele frequencies, the object `freq.hap` is available in the R package:

```
> str(freq.hap)

'data.frame':       20 obs. of  3 variables:
 $ key : int  1 1 2 2 2 2 2 2 2 3 ...
 $ hap : int  1 2 1111 1112 1121 1221 1222 2112 2222 11 ...
 $ freq: num  0.8 0.2 0.3373 0.2059 0.0024 ...

> freq.hap[1:6,]

  key  hap          freq
1   1    1 0.800000000
2   1    2 0.200000000
3   2 1111 0.337339745
4   2 1112 0.205929486
5   2 1121 0.002403846
6   2 1221 0.368589742
```

We can now impute the missing genotypes using these underlying haplotype frequencies.

```
> trio.tmp <- trio.check(dat=trio.gen2, is.linkage=FALSE)
> set.seed(123456)
> trio.imp3 <- trio.prepare(trio.dat=trio.tmp, freq=freq.hap, logic=FALSE)
> str(trio.imp3, max=1)

List of 3
 $ trio:'data.frame':       300 obs. of  12 variables:
 $ miss:'data.frame':       250 obs. of  5 variables:
 $ freq:'data.frame':       20 obs. of  3 variables:
 - attr(*, "class")= chr "trioPrepare"

> trio.gen2[1:6,]

  famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1 10001   1    0    0    1    1    1    1    0    0    0     0
2 10001   2    0    2   NA    0    2    0    0    0    0     0
3 10001   3    0    1    1    1    1    1    0   NA    0     0
4 10002   1    0   NA    2   NA    0   NA    0    0    0     1
5 10002   2    0   NA    0    0    2    0    0    0    0     0
6 10002   3    0    1    1    1    1    1    0    0    0     0

> trio.imp3$trio[1:6,]

  famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
1 10001   1    0    0    1    1    1    1    0    0    0     0
2 10001   2    0    2    0    0    2    0    0    0    0     0
3 10001   3    0    1    1    1    1    1    0    0    0     0
```

```
4 10002   1    0    0    2    2    0    2    0    0    0    1
5 10002   2    0    2    0    0    2    0    0    0    0    0
6 10002   3    0    1    1    1    1    1    0    0    0    0
```

6   Trio Logic Regression

After having prepared a matrix suitable for a trio logic regression analysis, the function `trioLR` can be used to perform this analysis.

To ensure that the following examples are fast to run, we here only use 1000 iterations, i.e. the minimum number of iterations allowed, in the stochastic search algorithms, simulated annealing and MCMC, employed in trio logic regression. **In a real trio logic regression analysis, at least a few hundred thousands of iterations should be used, where the number of iterations in particular depends on the number of variables considered in the analysis.**

The parameter `iter` for setting the number of iterations along with several other control parameters for the two stochastic search algorithms and the logic tree, i.e. the logic expression, grown in a trio logic regression can be specified by employing the argument `control` of the function `trioLR`, which in turn should be specified via the function `lrControl`. Here, we set

```
> my.control <- lrControl(start=1, end=-3, iter=1000, output=-4)
```

where `start` and `end` are the starting and end temperature on log10-scale in simulated annealing, where the temperature governs how likely it is that the model proposed in an iteration of trio logic regression is accepted although it acutally performs worse than the model accepted in the previous iteration (at the beginning, this acceptance probability is comparatively high so that many models are visited/accepted, whereas in the last iterations, this probability is very low so that virtually only models are accepted if they are better than the current trio logic regression model). Again, it is very hard to give

a good recommendation how to choose `start` and `end` (if they are not specified, the algorithm tries to find good choices for these two parameters, which, however, might works suboptimal), as their optimal specification highly depends on the data at hand. In our experience, `start = 1` and `end = -3` are often reasonable choices for a trio logic regression, but for a particular case-parent trio data set other choices might work better. The help page for the function `logreg.anneal.control` in the R package `LogicReg` gives a comprehensive introduction on how the control parameters of simulated annealing can be chosen.

Finally, we have set `output = -4`, so that the models visited during the search for the best trio logic regression model with an MCMC algorithm are not stored in file, but all statistics for individual SNPs, pairs of SNPs, and triplets of SNPs are computed.

Employing this specification of these four parameters and the defaults for the other control parameters, `trioLR` can be directly applied to the output of `trio.prepare`, i.e. for example `trio.bin` (see Section 5.2), by

```
> lr.out <- trioLR(trio.bin, control=my.control, rand=9876543)
> lr.out


          Trio Logic Regression

Search Algorithm: Simulated Annealing

A single model has been fitted:
-2.54 * ((snp6.D | snp2.R) & !snp4.D)
Score:  0.328
```

where we specify the argument `rand` to set the random number generator in a reproducible state. By default, the logic expression found by trio logic regression is printed as it has been found. Alternatively, it can also be printed in disjunctive normal form

```
> print(lr.out, asDNF=TRUE)


          Trio Logic Regression

Search Algorithm: Simulated Annealing
```

```
A single model has been fitted:
-2.54 * ((snp2.R & !snp4.D) | (!snp4.D & snp6.D))
Score:  0.328
```

This has the advantage that the interactions – in particular, in a statistical sense – contained in the logic expression are directly given by the AND-combinations (&) in this disjunctive normal form. In the example, this advantage admittedly does not really exist, but when a much larger and more complex logic tree is grown, the logic expression represented by this logic tree can be very hard to interpret.

It is also possible to consruct the matrix x containing the values of the cases and the matched pseudo-controls and the vector y comprising the class labels of the (artificial) subjects without using trio.prepare. In this situation, each column of x must represent one of the logic/binary variable, coded by zeros and ones, and each row must correspond to a case or a pseudo-control, where each block of four consecutive rows has to consist of the data for a case and the three matched pseudo-controls (in this order). In y, each case must be coded by a 3 and each pseudo-control by a 0 so that y is given by

```
> n.trios <- 100
> y <- rep(c(3, 0, 0, 0), n.trios)
```

where n.trios is the number of case-parent trios considered in a study. This number is here set to 100, as our example data set trio.bin consists of 100 trios. To avoid the construction of the matrix x, we extract this matrix for this example from trio.bin.

```
> x <- trio.bin$bin[,-1]
```

The same trio regression analysis as above can then be performed by

```
> lr.out2 <- trioLR(x, y, control=my.control, rand=9876543)
> lr.out2

        Trio Logic Regression

Search Algorithm: Simulated Annealing

A single model has been fitted:
-2.54 * ((snp6.D | snp2.R) & !snp4.D)
Score:  0.328
```

The trio logic regression model resulting from such an application can then be tested by a null-model permutation procedure, which checks whether there is signal in the data, or by a conditional permutation test for model selection. Both can be performed using the function `trio.permTest` in which the argument `conditional` specifies which of these permutation tests are done. Thus, the null-model permutation test with `n.perm = 20` permutations can be applied to the case-parent trio data in `trio.bin` by

```
> trio.permTest(lr.out, n.perm=20)
```

while the conditional permutation test can be performed by

```
> trio.permTest(lr.out, conditional=TRUE, n.perm=20)
```

By default, (trio) logic regression uses simulated annealing as search algorithm, and `trioLR` fits one trio logic regression model comprising one logic tree with a maximum of five leaves, i.e. five binary variables coded by zeros and ones Thus, the argument `nleaves` is by default set to `nleaves = 5`, whereas the number of logic trees cannot be changed in a trio logic regression.

Alternatively, several trio logic regression models with different maximum numbers of leaves/variables can be fitted by setting `nleaves` to a vector of length 2, where the first element specifies the lowest maximum number, and the second element the largest maximum number. If thus, for example, trio logic regression models should be fitted in which the maximum number of leaves varies between 3 and 5, then this can be done by

```
> lr.out3 <- trioLR(trio.bin, nleaves=c(3,5), control=my.control, rand=9876543)
> lr.out3

        Trio Logic Regression

Search Algorithm: Simulated Annealing

3 models have been fitted:
```

Logic Tree
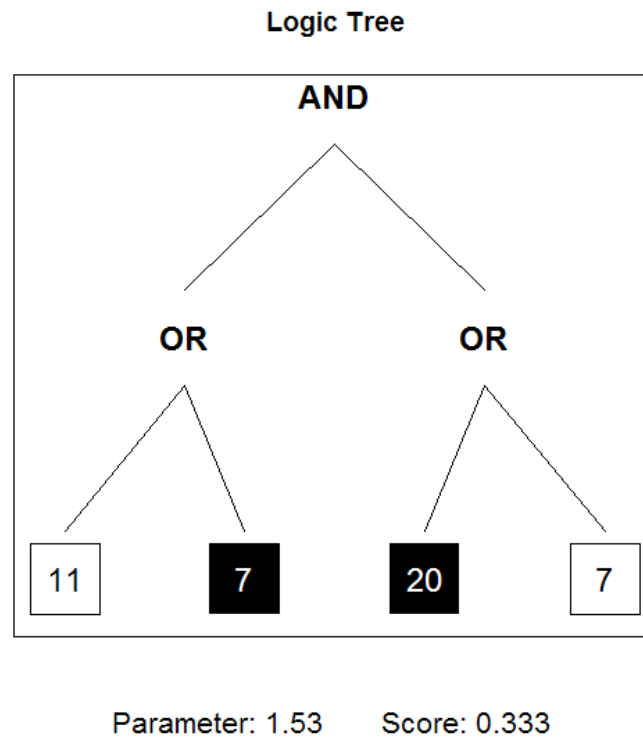

Parameter: 1.53   Score: 0.333

Figure 1: Logic tree built in a trio logic regression analysis of case-parent trio data in which a maximum of five leaves was allowed.

```
Model with a maximum of 3 leaves:
2.54 * ((!snp2.R & !snp6.D) | snp3.D)
Score: 0.328

Model with a maximum of 4 leaves:
-2.08 * ((!snp8.D | !snp7.D) | snp4.R)
Score: 0.327

Model with a maximum of 5 leaves:
1.53 * ((snp6.D | !snp4.D) & (!snp10.R | snp4.D))
Score: 0.333
```

The resulting logic trees cannot only be printed, but also plotted. If, for example, the logic tree in the third trio logic regression model (i.e. the model with a maximum of five leaves) should be plotted, then this can be done by

```
plot(lr.out3, whichTree=3)
```

(see Figure 1), where the argument `whichTree` needs only to be specified if several models have been fitted.

Alternatively to simulated annealing, a greedy search can be employed in trio logic regression by changing the argument `search` to `"greedy"`.

```
> lr.out4 <- trioLR(trio.bin, search="greedy", rand=9876543)
> lr.out4

          Trio Logic Regression

Search Algorithm: Greedy

6 models have been fitted:

Model with 0 leaves:
Model contains no variables.
Score: 0.347

Model with a maximum of 1 leaf:
0.762 * snp9.D
Score: 0.343

Model with a maximum of 2 leaves:
1.43 * (snp9.D & !snp3.R)
Score: 0.336

Model with a maximum of 3 leaves:
1.98 * ((snp9.D & snp7.D) & !snp3.R)
Score: 0.33

Model with a maximum of 4 leaves:
2.1 * ((snp9.D & (snp7.D | !snp8.D)) & !snp3.R)
Score: 0.325

Model with a maximum of 5 leaves:
2.43 * ((snp9.D & (snp7.D | !snp8.D)) & (!snp3.R & !snp6.R))
Score: 0.321
```

While trio logic regression based on simulated annealing or a greedy search tries to identify the logic expression, and thus, the trio logic regression model, that provides the best prediction for the disease risk, the main goal of trio MC logic regression, i.e. trio logic regression based on MCMC is the specification of the individual relevance of SNPs and the joint importance of pairs and triplets of SNPs for disease risk by counting how often

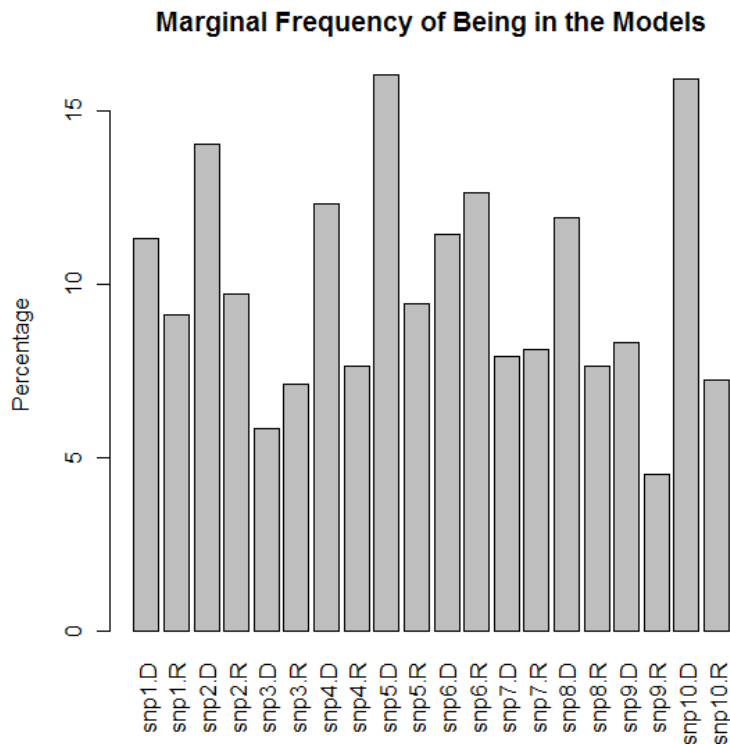**Marginal Frequency of Being in the Models**

Figure 2: Visualization of the results of a trio MC logic regression analysis for the individual SNPs.

the individual SNPs occur in the models visited (and accepted) during the MCMC search and how ofteh pairs and triplets of SNPs occur jointly in these models (cf. Kooperberg and Ruczinski, 2005). This trio MC logic regression can be performed by

```
> lr.out5 <- trioLR(trio.bin, search="mcmc", control=my.control, rand=9876543)
> lr.out5

        Trio Logic Regression

Search Algorithm: MCMC

Visited models are not stored.
```

The results of this analysis can be visualized by plotting for each SNP the percentages of visited models in which this SNP occurs (freqType = 1; the default), and for each pair of SNPs in how many of the visited models this pair cocurs jointly (freqType = 2)

Figure 3: Values of the importance measure of trioFS for the five most important identified SNP interactions.

and the observed-to-expected ratio of being jointly in the models (`freqType = 3`). For example, in Figure 2,

```
plot(lr.out5, freqType=1, useNames=TRUE)
```

the frequencies for the individual SNPs are displayed.

Another way to quantify the importance of SNPs and SNP interactions for the prediction of the disease risk is to apply trio logic regression to several subsets of the case-parent trio data and then employ the resulting trio logic regression models to compute importance measures for the interactions found. This procedure called trioFS (trio Feature Selection) is implemented in the function `trioFS` that can be used in a similar way as `trioLR`. For example, trioFS can be applied to the case-parent trio data by

36

```
> fs.out <- trioFS(trio.bin, B=5, control=my.control, rand=9876543)
> fs.out

Identification of Interactions Using Trio Logic Regression

Number of Iterations:  5
Sampling Method:       Bagging
Number of Trees:       1
Max. Number of Leaves: 5

The 5 Most Important Interactions:

  Importance Proportion                Expression
1     13.369          0.20 !snp3.R & !snp6.R & snp9.D
2      0.621          0.20 !snp2.D & snp5.D & !snp8.R
3      0.028          0.20  snp5.R & !snp7.R & snp9.D
4      0.000          0.20 snp1.R & !snp2.D & !snp6.R
5      0.000          0.20 !snp2.D & snp5.D & !snp6.R
```

where we here consider only B = 5 subsets of the data, while in a real analysis, at least B = 20 subsets should be considered. By default, simulated annealing is used in this search. Alternatively, it is also possible to do a greedy search by setting the argument `fast` of `trioFS` to TRUE.

The importances of the identified SNP interactions cannot only be printed, but also be plotted.

```
plot(fs.out)
```

(see Figure 3).

7 Detection of LD Blocks

For the estimation of the haplotype structure that might be used in the R function `trio.prepare`, this package also includes functions for the fast computation of the pairwise D' and r^2 values for hundreds or thousands of SNPs, and for the identification of

LD blocks in these genotype data using a modified version of the algorithm proposed by Gabriel *et al.* (2002). For the latter, it is assumed that the SNPs are ordered by their position on the chromosomes.

These functions are not restricted to trio data, but can also be applied to population-based data. The only argument of these functions specifically included for trio data is parentsOnly. If set to TRUE, only the genotypes of the parents are used in the determination of the pairwise values of the LD measures and the estimation of the LD blocks. Furthermore, each parent is only considered once so that parents with more than one offspring do not bias the estimations. If trio data is used as input, the functions assume that the matrix containing the SNP data is in genotype format.

Here, we consider a simulated matrix LDdata from a population-based study. Thus, all subjects are assumed to be unrelated. This matrix contains simulated genotype data for 10 LD blocks each consisting of 5 SNPs each typed on 500 subjects. The pairwise D' and r^2 values for the SNPs in this matrix can be computed by

```
> ld.out <- getLD(LDdata, asMatrix=TRUE)
```

where by the default these values are stored in vectors to save memory. If asMatrix is set to TRUE, the values will be stored in matrices. The pairwise LD values for the first 10 SNPs (rounded to the second digit) can be displayed by

```
> round(ld.out$Dprime[1:10,1:10], 2)

      S1   S2   S3   S4   S5   S6   S7   S8 S9 S10
S1    NA   NA   NA   NA   NA   NA   NA   NA NA  NA
S2  0.99   NA   NA   NA   NA   NA   NA   NA NA  NA
S3  0.98 1.00   NA   NA   NA   NA   NA   NA NA  NA
S4  0.98 0.99 1.00   NA   NA   NA   NA   NA NA  NA
S5  0.97 0.98 0.99 1.00   NA   NA   NA   NA NA  NA
S6  0.09 0.06 0.05 0.05 0.04   NA   NA   NA NA  NA
S7  0.11 0.09 0.08 0.08 0.07 0.99   NA   NA NA  NA
S8  0.13 0.11 0.10 0.10 0.09 0.99 1.00   NA NA  NA
S9  0.14 0.11 0.10 0.11 0.10 0.99 1.00 1.00 NA  NA
S10 0.16 0.13 0.11 0.12 0.11 0.97 0.98 0.98  1  NA

> round(ld.out$rSquare[1:10,1:10], 2)
```

```
       S1   S2   S3   S4 S5   S6   S7   S8   S9 S10
S1     NA   NA   NA   NA NA   NA   NA   NA   NA   NA
S2   0.97   NA   NA   NA NA   NA   NA   NA   NA   NA
S3   0.94 0.97   NA   NA NA   NA   NA   NA   NA   NA
S4   0.93 0.96 1.00   NA NA   NA   NA   NA   NA   NA
S5   0.91 0.94 0.98 0.98 NA   NA   NA   NA   NA   NA
S6   0.00 0.00 0.00 0.00  0   NA   NA   NA   NA   NA
S7   0.00 0.00 0.00 0.00  0 0.97   NA   NA   NA   NA
S8   0.00 0.00 0.00 0.00  0 0.95 0.98   NA   NA   NA
S9   0.00 0.00 0.00 0.00  0 0.93 0.96 0.98   NA   NA
S10  0.00 0.00 0.00 0.00  0 0.91 0.94 0.96 0.98   NA
```

and the pairwise LD plot for all SNPs can be generated by

```
> plot(ld.out)
```

(see Figure 4). This figure shows the r^2-values. The D' values can be plotted by

```
> plot(ld.out, "Dprime")
```
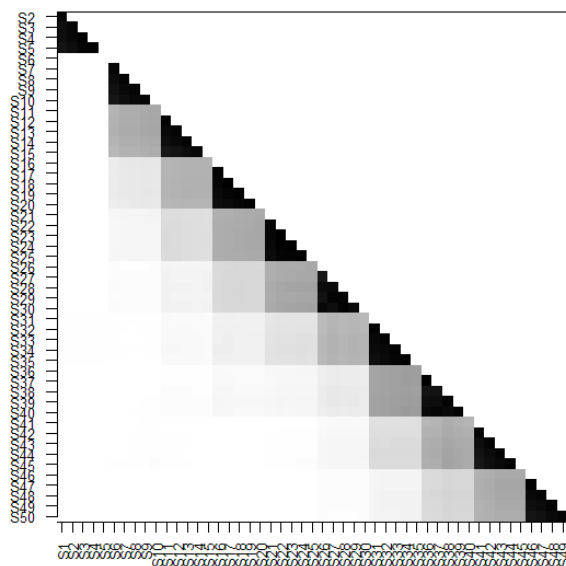
(not shown).

Figure 4: Pairwise r^2 values for the SNPs from LDdata.

The LD blocks in genotype data can be identified using the modified algorithm of Gabriel *et al.* (2002) by calling

```
> blocks <- findLDblocks(LDdata)
> blocks

Found 10 LD blocks containing between 5 and 5 SNPs.
0 of the 50 SNPs do not belong to a LD block.

Used Parameter:
Strong LD:     C_L >= 0.7 and C_U >= 0.98
Recombination: C_U < 0.9
(C_L and C_U are the lower and upper bound of
the 90%-confidence intervals for D')
LD blocks: Ratio >= 9
```

Alternatively, the output of `getLD` can be used when `addVarN` has been set to `TRUE` in `getLD` to store additional information on the pairwise LD values.

```
> ld.out2 <- getLD(LDdata, addVarN=TRUE)
> blocks2 <- findLDblocks(ld.out2)
> blocks2

Found 10 LD blocks containing between 5 and 5 SNPs.
0 of the 50 SNPs do not belong to a LD block.

Used Parameter:
Strong LD:     C_L >= 0.7 and C_U >= 0.98
Recombination: C_U < 0.9
(C_L and C_U are the lower and upper bound of
the 90%-confidence intervals for D')
LD blocks: Ratio >= 9
```

The blocks can also be plotted by

```
> plot(blocks)
```

(see Figure 5). In this figure, the borders of the LD blocks are marked by red lines. By default, the three categories used by the algorithm of Gabriel *et al.* (2002) to define the LD blocks are displayed. Since this algorithm is based on the D' values, it is also possible to show these values in the LD (block) plot.
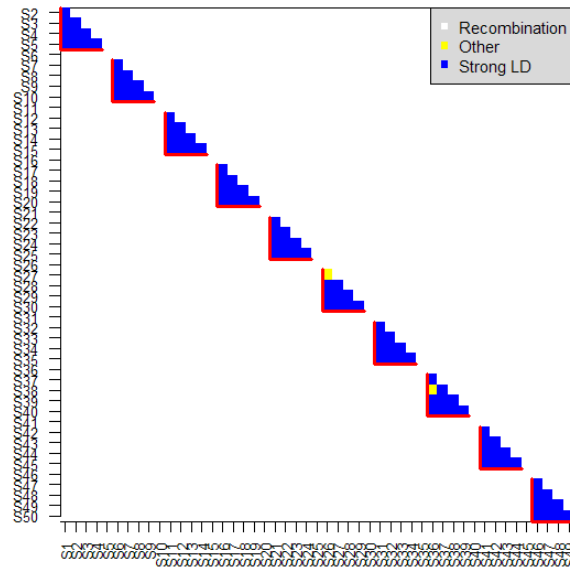
Figure 5: LD blocks as found by the modified algorithm of Gabriel *et al.* (2002). The borders of the LD blocks are marked by red lines. The color for the LD between each pair of SNPs is defined by the three categories used by Gabriel *et al.* (2002) to define the LD blocks.
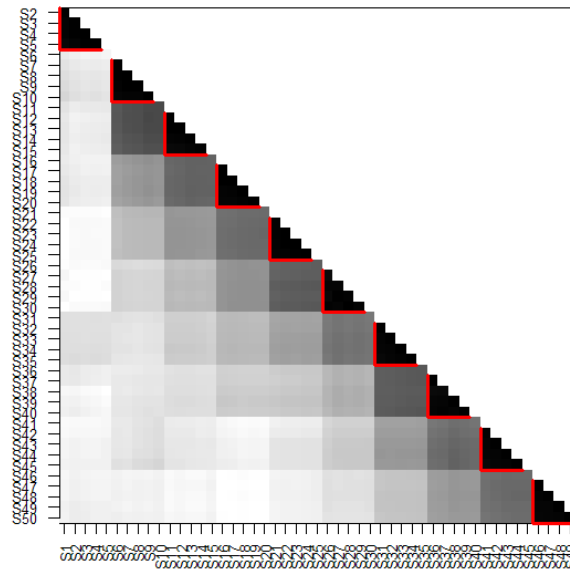

Figure 6: LD blocks as found by the modified algorithm of Gabriel *et al.* (2002). The borders of the LD blocks are marked by red lines. The darker the field for each pair of SNPs, the larger is the D' value for the corresponding SNP pair.

```
> plot(blocks, "Dprime")
```

(see Figure 6).

As mentioned in Section 5, the haplotype structure required by `trio.prepare` can be obtained by

```
> hap <- as.vector(table(blocks$blocks))
> hap

 [1] 5 5 5 5 5 5 5 5 5 5
```

8   Simulation

The function `trio.sim` simulates case-parents trio data when the disease risk of children is specified by (possibly higher-order) SNP interactions. The mating tables and the respective sampling probabilities depend on the haplotype frequencies (or SNP minor allele frequencies when the SNP does not belong to a block). This information is specified in the `freq` argument of the function `trio.sim`. The probability of disease is assumed to be described by the logistic term $\mathrm{logit}(p) = \alpha + \beta \times \text{Interaction}$, where $\alpha = \mathrm{logit}\,(\texttt{prev}) = \log(\frac{\texttt{prev}}{1-\texttt{prev}})$ and $\beta = \log(\texttt{OR})$. The arguments `interaction`, `prev` and `OR`, are specified in the function `trio.sim`. Generating the mating tables and the respective sampling probabilities, in particular for higher order interactions, can be very CPU and memory intensive. We show how this information, once it has been generated, can be used for future simulations, and thus, speed up the simulations dramatically.

8.1   A Basic Example

We use the built-in object `simuBkMap` in a basic example to show how to simulate case-parent trios when the disease risk depends on (possibly higher order) SNP interactions. This file contains haplotype frequency information on 15 blocks with a total of 45 loci. In this example, we specify that the children with two variant alleles on SNP1 and two

variant alleles on SNP5 have a higher disease risk. We assume that `prev = 0.001` and `OR = 2` in the logistic model specifying disease risk, and simulate a single replicate of 20 trios total.

```
> str(simuBkMap)

'data.frame':       66 obs. of  3 variables:
 $ key : Factor w/ 15 levels "10-1","10-10",..: 1 1 1 8 8 8 8 9 9 9 ...
 $ hap : int  11 21 22 121 122 111 222 21 22 12 ...
 $ freq: num  0.099 0.228 0.673 0.006 0.026 0.1 0.867 0.079 0.441 0.48 ...

> simuBkMap[1:7,]

    key hap  freq
1 10-1  11 0.099
2 10-1  21 0.228
3 10-1  22 0.673
4 10-2 121 0.006
5 10-2 122 0.026
6 10-2 111 0.100
7 10-2 222 0.867

> sim <- trio.sim(freq=simuBkMap, interaction="1R and 5R", prev=.001, OR=2,
+ n=20, rep=1)
> str(sim)

List of 1
 $ : num [1:60, 1:47] 1 1 1 2 2 2 3 3 3 4 ...
   ..- attr(*, "dimnames")=List of 2
   .. ..$ : NULL
   .. ..$ : chr [1:47] "famid" "pid" "snp1" "snp2" ...

> sim[[1]][1:6, 1:12]

      famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
[1,]      1   1    2    2    2    2    2    0    2    2    0     1
[2,]      1   2    2    2    2    2    2    1    2    2    1     0
[3,]      1   3    2    2    2    2    2    0    2    2    1     1
[4,]      2   1    1    1    2    2    2    0    2    2    1     1
[5,]      2   2    2    1    2    2    2    2    2    1    0     2
[6,]      2   3    1    1    2    2    2    1    2    2    0     2
```

8.2 Using Estimated Haplotype Frequencies

In this example we estimate the haplotype frequencies in the built-in data set `trio.gen1`, which contains genotypes for 10 SNPs in 100 trios. These estimated frequencies are then

used to simulate 20 trios for the above specified disease risk model.

```
> trio.tmp <- trio.check(dat=trio.gen1, is.linkage=FALSE)
> trio.impu <- trio.prepare(trio.dat=trio.tmp, blocks=c(1,4,2,3), logic=TRUE)
> str(trio.impu, max=2)

List of 3
 $ bin : num [1:400, 1:21] 3 0 0 0 3 0 0 0 3 0 ...
  ..- attr(*, "dimnames")=List of 2
 $ miss: NULL
 $ freq:'data.frame':        19 obs. of  3 variables:
  ..$ key :Class 'AsIs'  chr [1:19] "ch-1" "ch-1" "ch-h2" "ch-h2" ...
  ..$ hap : num [1:19] 1 2 1111 1112 1121 ...
  ..$ freq: num [1:19] 0.9425 0.0575 0.325 0.2225 0.0075 ...
 - attr(*, "class")= chr "trioPrepare"

> trio.impu$freq[1:7,]

    key  hap           freq
1  ch-1    1 9.425000e-01
2  ch-1    2 5.750000e-02
3 ch-h2 1111 3.250000e-01
4 ch-h2 1112 2.225000e-01
5 ch-h2 1121 7.500000e-03
6 ch-h2 1221 3.350000e-01
7 ch-h2 1222 3.665409e-09

> sim <- trio.sim(freq=trio.impu$freq, interaction="1R and 5R", prev=.001, OR=2,
+ n=20, rep=1)
> str(sim)

List of 1
 $ : num [1:60, 1:12] 1 1 1 2 2 2 3 3 3 4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:12] "famid" "pid" "snp1" "snp2" ...

> sim[[1]][1:6, ]

     famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
[1,]     1   1    0    0    1    1    0    0    1    1    1     0
[2,]     1   2    1    0    1    1    0    2    0    0    0     0
[3,]     1   3    0    0    2    2    0    1    1    1    1     0
[4,]     2   1    0    0    2    2    0    0    0    0    0     0
[5,]     2   2    0    0    1    1    1    0    1    0    0     0
[6,]     2   3    0    0    2    2    0    0    0    0    0     0
```

As before, the object containing the haplotype frequency information can also be generated from external haplotype frequencies and SNP minor allele frequencies. In the following

example we specify the haplotype frequencies, and generate two replicates of ten trios each.

```
> sim <- trio.sim(freq=freq.hap, interaction="1R or 4D", prev=.001, OR=2,
+ n=10, rep=2)
> str(sim)

List of 2
 $ : num [1:30, 1:12] 1 1 1 2 2 2 3 3 3 4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:12] "famid" "pid" "snp1" "snp2" ...
 $ : num [1:30, 1:12] 1 1 1 2 2 2 3 3 3 4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:12] "famid" "pid" "snp1" "snp2" ...

> sim[[1]][1:6,]

      famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
[1,]      1   1    0    0    1    1    1    0    0    0    0     0
[2,]      1   2    1    1    1    1    1    0    0    0    0     0
[3,]      1   3    0    1    0    0    2    0    0    0    0     0
[4,]      2   1    0    0    0    0    0    2    1    0    0     0
[5,]      2   2    1    1    1    1    1    0    0    1    0     0
[6,]      2   3    0    1    0    0    1    1    0    1    0     0
```

8.3   Using Step-Stones

Generating the mating tables and the respective sampling probabilities necessary to simulate case-parent trios can be very time consuming for interaction models involving three or more SNPs. In simulation studies, many replicates of similar data are usually required, and generating these sampling probabilities in each instance would be a large and avoidable computational burden (CPU and memory). The sampling probabilities depend foremost on the interaction term and the underlying haplotype frequencies, and as long as these remain constant in the simulation study, the mating table information and the sampling probabilities can be "recycled." This is done by storing the relevant information (denoted as "step-stone") as a binary R file in the working directory, and loading the binary file again in future simulations, speeding up the simulation process dramatically. It is even possible

to change the parameters `prev` and `OR` in these additional simulations, as the sampling probabilities can be adjusted accordingly.

In the following example, we first simulate case-parent trios using a three-SNP interaction risk model, and save the step-stone object. We then simulate additional trios with a different parameter `OR`, using the previously generated information.

```
> sim <- trio.sim(freq=freq.hap, interaction="1R or (6R and 10D)", prev=.001,
+ OR=2, n=10, rep=1)
> str(sim)

List of 1
 $ : num [1:30, 1:12] 1 1 1 2 2 2 3 3 3 4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:12] "famid" "pid" "snp1" "snp2" ...

> sim[[1]][1:6,]

     famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
[1,]     1   1    0    1    1    1    1    0    0    0    0     1
[2,]     1   2    1    1    1    1    1    1    0    0    1     0
[3,]     1   3    0    1    1    1    1    0    0    0    0     0
[4,]     2   1    1    0    2    2    0    0    1    0    0     1
[5,]     2   2    0    0    0    0    1    1    0    0    0     1
[6,]     2   3    1    0    1    1    0    0    1    0    0     1

> sim <- trio.sim(freq=freq.hap, interaction="1R or (6R and 10D)", prev=.001,
+ OR=3, n=10, rep=1, step.save="step3way")
> str(sim, max=1)

List of 1
 $ : num [1:30, 1:12] 1 1 1 2 2 2 3 3 3 4 ...
  ..- attr(*, "dimnames")=List of 2

> sim[[1]][1:6,]

     famid pid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 snp9 snp10
[1,]     1   1    0    0    0    0    1    0    0    0    0     0
[2,]     1   2    0    0    1    1    0    0    1    0    0     1
[3,]     1   3    0    0    1    1    1    0    1    0    0     1
[4,]     2   1    0    0    1    1    1    0    0    0    0     0
[5,]     2   2    1    0    1    1    1    1    0    0    0     0
[6,]     2   3    0    0    2    2    0    0    0    0    0     0
```

Acknowledgments

References

Cordell, H.J. (2002). Epistasis: What it Means, what it Doesn't Mean, and Statistical Methods to Detect it in Humans. *Human Molecular Genetics*, **11**, 2463–2468.

Cordell, H.J., Barratt, B.J., and Clayton, D.G. (2004). Case/Pseudocontrol Analysis in Genetic Association Studies: A Unified Framework for Detection of Genotype and Haplotype Associations, Gene-Gene and Gene-Environment Interactions, and Parent-of-Origin Effects. *Genetic Epidemiology*, **26**, 167–185.

Gabriel, S.B., Schaffner, S.F., Nguyen, H., Moore, J.M., Roy, J., Blumenstiel, B., Higgins, J., DeFelice, M., Lochner, A., Faggart, M., Liu-Cordero, S.N., Rotimi, C., Adeyemo, A., Cooper, R., Ward, R., Lander, E.S., Daly, M.J., and Altshuler, D. (2002). The Structure of Haplotype Blocks in the Human Genome. *Science*, **296**, 2225–2229.

Kooperberg, C. and Ruczinski, I. (2005). Identifying Interacting SNPs Using Monte Carlo Logic Regression. *Genetic Epidemiology*, **28**, 157–170.

Li, Q., Fallin, M.D., Louis, T.A., Lasseter, V.K., McGrath, J.A., Avramopoulos, D., Wolyniec, P.S., Valle, D., Liang, K.Y., Pulver, A.E., and Ruczinski, I. (2010). Detection of SNP-SNP Interactions in Trios of Parents with Schizophrenic Children. *Genetic Epidemiology*, **34**, 396–406.

Schaid, D.J. (1996). General Score Tests for Associations of Genetic Markers with Disease Using Cases and Their Parents. *Genetic Epidemiology*, **13**, 423–449.

Schwender, H., Bowers, K., Fallin, M.D., and Ruczinski, I. (2011). Importance Measure for Epistatic Interactions in Case-Parent Trios. *Annals of Human Genetics*, **75**, 122–132.

Schwender, H., Taub, M.A., Beaty, T.H., Marazita, M.L., and Ruczinski, I. (2011). Rapid Testing of SNPs and Gene-Environment Interactions in Case-Parent Trio Data Based on Exact Analytic Parameter Estimation. *Biometrics*. DOI: 10.1111/j.1541-0420.2011.01713.x.

Spielman, R.S., McGinnis, R.E., and Ewens, W.J. (1993). Trsnmmission Test for Linkage Disequilibrium: The Insulin Gene Region and Insulin-Dependent Diabetes Mellitus (IDDM). *American Journal of Human Genetics*, **52**, 506–516.