A Vignette for the R Package `trio`

1 Introduction

The R package, `trio`, provides functionality relevant for the analysis of case-parent trio data with *trio logic regression*. Two major features are implemented in this package: 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 a framework that allows for the simulation of case-parent data where the risk of disease is specified by (higher order) SNP-SNP interactions.

The first section of this vignette is devoted to the steps relevant for data processing, to derive a matrix or data frame suitable as input for trio logic regression, starting from a linkage or genotype file which possibly contains 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 available.

The second section of the vignette explains in more detail how to set up simulations of case-parent data where the risk of disease is specified by SNP-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 data are generated.

2 Generating data for trio logic regression input

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 in either in linkage or genotype format, see below). If no Mendelian inconsistencies are detected, this function creates an object that is passed to the function `trio`. 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 4). Given the LD block lengths, the haplotype frequencies can be estimated, using the function `haplo.em()` in the `haplo.stat` package.

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 `trio` package requires that the trio data are already available as an R object, either as linkage file in ped format (the default), or as genotype file. The first six columns in such a 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 such a data set. 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 SNPs is encoded in two variables denoting the alleles.

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

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.

```
> 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   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
```

Taking the SNP LD structure 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 4).

The function `trio()`, 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()` 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.

```
> trio.bin = trio(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:

> 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      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,]      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
```

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` is 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()`.

```
> data(trio.gen1)
> 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=F)
> trio.bin = trio(trio.dat=trio.tmp, blocks=c(1,4,2,3))
```

Missing genotype information

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

```
> data(trio.ped2)
> 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()` 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.

```
> trio.bin = trio(trio.dat=trio.tmp, blocks=c(1,4,2,3))
> 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      0      0      1      0      1      0      1      0
[3,] 0      0      0      0      0      0      0      0      0      1      1
[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,]      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       1       0
[8,]      1      0      0      0      0      0      0      0       0       0
```

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

```
> data(trio.gen2)
> 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=F)
> trio.bin = trio(trio.dat=trio.tmp, blocks=c(1,4,2,3))
> 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      1      0      1      0      1      0
[3,] 0      0      0      1      0      0      0      0      0      1      1
[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,]      1      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,]      0      0      0      0      0      0      0      0       0       0
[7,]      1      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()` also allows for this option by using the argument `logic=F`. In the resulting object, the matrix `bin` is then replaced by the data frame `trio`, and `miss` and `freq` are also returned.

```
> data(trio.gen2)
> trio.tmp = trio.check(dat=trio.gen2, is.linkage=F)
> trio.imp = trio(trio.dat=trio.tmp, blocks=c(1,4,2,3), logic=F)
> 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:

> 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
```

```
> print(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
```

```
> print(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    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
```

The same applies to pedigree data:

```
> data(trio.ped2)
> trio.tmp = trio.check(dat=trio.ped2)
> trio.imp = trio(trio.dat=trio.tmp, blocks=c(1,4,2,3), logic=F)
```

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.

```
> data(trio.ped.err)
> 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 straightforward 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=T` 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=T)
> 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.

```
> data(trio.gen.err)
> trio.tmp = trio.check(dat=trio.gen.err, is.linkage=F)

[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=F, replace=T)
> 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
```

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` 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:

```
> data(freq.hap)
> 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 0.33734 0.20593 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.

```
> data(trio.gen2)
> trio.tmp = trio.check(dat=trio.gen2, is.linkage=F)
> trio.imp = trio(trio.dat=trio.tmp, freq=freq.hap, logic=F)
> 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':        20 obs. of  3 variables:

> print(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
```

```
> print(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
```

3 Simulation

The function `trio.sim()` simulates case-parents trio data when the disease risk of children is specified by (possibly higher-order) SNP-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 logit$(p) = \alpha + \beta \times$ Interaction, where $\alpha =$ 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.

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-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.

```
> data(simuBkMap)
> 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    1    1    2    2    2    2    2    1    2     1
```

```
[2,]    1   2    1    1    1    1    1    2    1    2    2     0
[3,]    1   3    2    2    2    2    2    2    2    1    2     0
[4,]    2   1    2    1    2    2    2    1    2    2    1     1
[5,]    2   2    1    1    2    2    2    1    2    2    1     1
[6,]    2   3    2    2    2    2    2    0    2    2    2     0
```

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.

```
> data(trio.gen1)
> trio.tmp = trio.check(dat=trio.gen1, is.linkage=F)
> trio.impu = trio(trio.dat=trio.tmp, blocks=c(1, 4, 2, 3), logic=T)
> 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 ...

> 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 3.522628e-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    0    0    0    2    0    0    0     0
[2,]     1   2    0    1    0    0    2    0    0    0    0     1
[3,]     1   3    0    1    0    0    1    1    0    0    0     1
[4,]     2   1    0    0    1    1    0    2    0    0    0     1
[5,]     2   2    0    1    1    1    1    1    0    0    0     1
[6,]     2   3    0    0    1    1    0    1    0    0    0     1
```

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.

```
> data(freq.hap)
> 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    2    0    2    2    0    1    0    0    0     1
[2,]     1   2    0    1    1    1    1    1    0    0    0     1
[3,]     1   3    1    1    1    1    1    0    0    0    0     1
[4,]     2   1    0    0    0    0    0    0    0    0    0     1
[5,]     2   2    0    0    0    0    0    2    1    0    0     0
[6,]     2   3    0    0    0    0    0    1    0    0    0     0
```

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.

```
> data(freq.hap)
> 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    0    0    0    1    2    0    0    0     0
[2,]     1   2    1    1    1    1    1    1    0    0    0     1
```

```
[3,]     1   3    0    1    0    0    2    2    0    0    0     1
[4,]     2   1    1    0    2    2    0    0    0    0    0     0
[5,]     2   2    0    0    0    0    1    1    0    0    0     0
[6,]     2   3    0    0    1    1    1    1    0    0    0     0
```

```
> 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    2    2    0    2    0    0    0     0
[2,]     1   2    0    0    0    0    2    0    0    1    1     1
[3,]     1   3    0    0    1    1    1    1    0    0    0     1
[4,]     2   1    0    0    2    2    0    1    2    0    0     1
[5,]     2   2    1    0    1    1    0    1    0    0    0     1
[6,]     2   3    0    0    2    2    0    1    1    0    0     0
```

4 Detection of LD blocks

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.

The matrix `LDdata` available in

```
> data(LDdata)
```

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 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 1). This figure shows the r^2-values. The D' values can be plotted by

```
plot(ld.out, "Dprime")
```

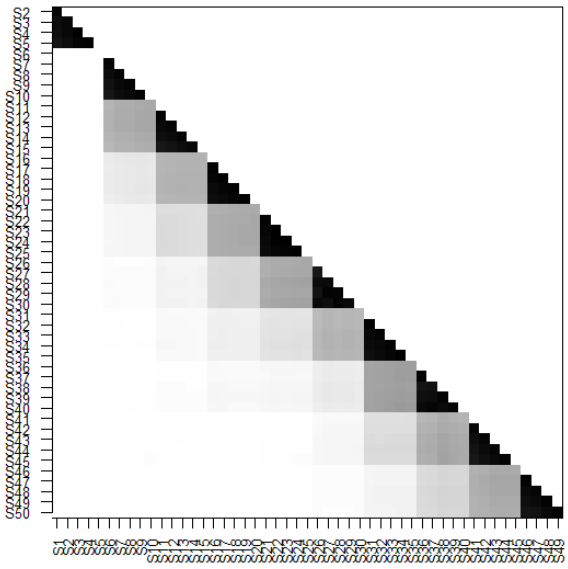

Figure 1: Pairwise r^2 values for the SNPs from LDdata.

(not shown).

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 2). 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.

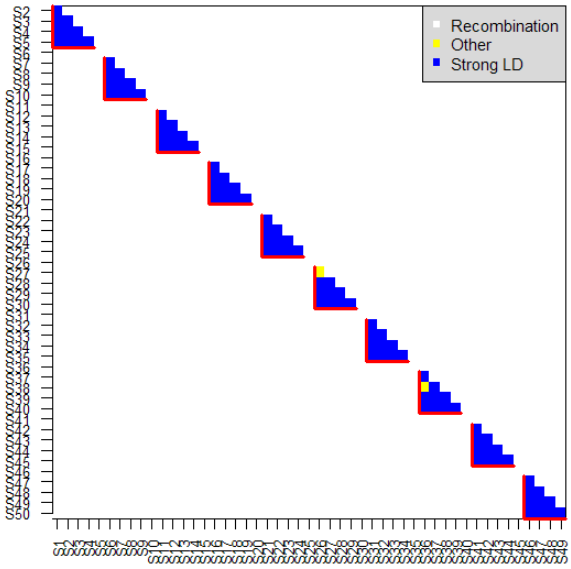

Figure 2: LD blocks as found by the modified algorithm of Gabriel et al. (2002). The borders of the LD blocks are marked by red lines. The color for the LD between each pair of SNPs is defined by the three categories used by Gabriel et al. (2002) to define the LD blocks.

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

(see Figure 3).

As mentioned in Section 2, the haplotype structure required by `trio` can be obtained by

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

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

Acknowledgments

Support was provided by NIH grants R01 DK061662 and R01 HL090577, and by DFG grant SCHW 1508/1-1.

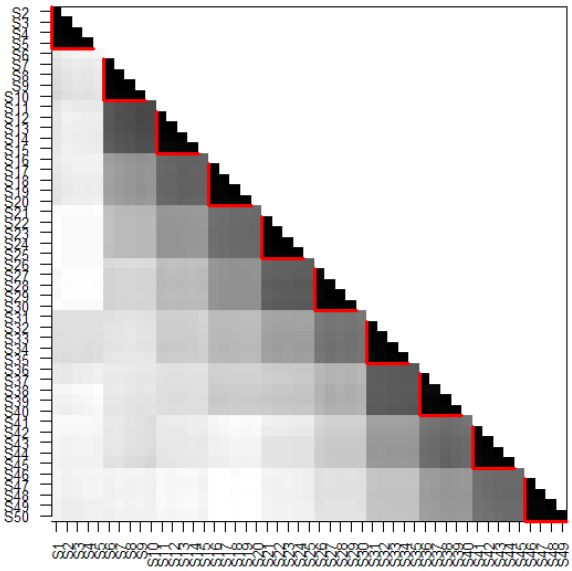

Figure 3: LD blocks as found by the modified algorithm of Gabriel et al. (2002). The borders of the LD blocks are marked by red lines. The darker the field for each pair of SNPs, the larger is the D' value for the corresponding SNP pair.

References

Gabriel, S. B., Schaffner, S. F., Nguyen, H., Moore, J. M., Roy, J., Blumenstiel, B., H., 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.