Haplo Stats
(version 1.5.0)

Statistical Methods for Haplotypes When Linkage Phase is Ambiguous

Jason P. Sinnwell* and Daniel J. Schaid
Mayo Clinic Division of Health Sciences Research
Rochester MN USA

December 5, 2011

*sinnwell@mayo.edu
6.7.3 Rank of Information Matrix and eps.svd (NEW) ..................... 36
6.7.4 Rare Haplotypes and haplo.min.info .............................. 39

7 Methods for haplo.glm (NEW) ........................................ 40
  7.1 fitted.values ..................................................... 41
  7.2 residuals .......................................................... 41
  7.3 vcov ................................................................. 41
  7.4 anova and Model Comparison .................................... 42

8 Extended Applications .................................................. 44
  8.1 Combine Score and Group Results: haplo.score.merge ............. 44
  8.2 Case-Control Haplotype Analysis: haplo.cc ....................... 44
  8.3 Score Tests on Sub-Haplotypes: haplo.score.slide ................ 47
  8.3.1 Plot Results from haplo.score.slide ........................ 48
  8.4 Scanning Haplotypes Within a Fixed-Width Window: haplo.scan ... 50
  8.5 Sequential Haplotype Scan Methods: seqhap ....................... 51
  8.5.1 Plot Results from seqhap .................................. 54
  8.6 Creating Haplotype Effect Columns: haplo.design ................ 56

9 License and Warranty .................................................... 59

10 Acknowledgements ..................................................... 59

A Counting Haplotype Pairs When Marker Phenotypes Have Missing Alleles 60
1 Introduction

Haplo Stats is a suite of R routines for the analysis of indirectly measured haplotypes. The statistical methods assume that all subjects are unrelated and that haplotypes are ambiguous (due to unknown linkage phase of the genetic markers), while also allowing for missing alleles.

The user-level functions in Haplo Stats are:

- **haplo.em**: for the estimation of haplotype frequencies and posterior probabilities of haplotype pairs for each subject, conditional on the observed marker data
- **haplo.glm**: generalized linear models for the regression of a trait on haplotypes, with the option of including covariates and interactions
- **haplo.score**: score statistics to test associations between haplotypes and a variety of traits, including binary, ordinal, quantitative, and Poisson
- **haplo.score.slide**: haplo.score computed on sub-haplotypes of a larger region
- **seqhap**: sequentially scan markers in enlarging a haplotype for association with a trait
- **haplo.cc**: run a combined analysis for haplotype frequencies, scores, and regression results for a case-control study
- **haplo.power.qt/haplo.power.cc**: power or sample size calculations for quantitative or binary trait
- **haplo.scan**: search for a trait locus for all sizes of sub-haplotypes within a fixed maximum window width for all markers in a region
- **haplo.design**: create a design matrix for haplotype effects

This manual explains the basic and advanced usage of these routines, with guidelines for running the analyses and interpreting results. We provide many of these details in the function help pages, which are accessed within an R session using `help(haplo.em)`, for example. We also provide brief examples in the help files, which can be run in the R session with `example(haplo.em)`.

1.1 Updates

This version of Haplo Stats contains updates to `haplo.glm` in section 6 and new methods written for it that resemble glm class methods. These methods include residuals, fitted.values, vcov, and anova, and they are detailed in section 7. For full history of updates see the NEWS file, or type `news(package="haplo.stats")` in the R command prompt.

1.2 Operating System and Installation

Haplo Stats version 1.5.0 is written for R (version 2.14.0). It has been uploaded to the Comprehensive R Archive Network (CRAN), and is made available on various operating systems through CRAN. Package installation within R is made simple from within R using `install.packages("haplo.stats")`, but other procedures for installing R packages can be found at the R project website (http://www.r-project.org).
1.3 R Basics

For users who are new to the R environment, we demonstrate some basic concepts. In the following example we create a vector of character alleles and use the `table` function to get allele counts. We first show how to save the results of `table(snp)` into an R session variable, `tab`. We show that `tab` is an object of the `table` class, and use the print and summary methods that are defined for `table` objects. Note that when you enter just `tab` or `table(snp)` at the prompt, the print method is invoked.

```r
> tab <- table(snp)
> tab

snp
A  T
3  4

> class(tab)
[1] "table"

> print.table(tab)

snp
A  T
3  4

> summary(tab)

Number of cases in table: 7
Number of factors: 1

> tab[2]

T
4

> table(snp)

snp
A  T
3  4
```

The routines in `haplo.stats` are computationally intensive and return lots of information in the returned object. Therefore, we assign classes to the returned objects and provide various methods for each of them.
2 Data Setup

We first show some typical steps when you first load a package and look for details on a function of interest. In the sample code below, we load haplo.stats, check which functions are available in the package, view a help file, and run the example that is within the help file.

> # load the library, load and preview at demo dataset
> library(haplo.stats)
> ls(name="package:haplo.stats")
> help(haplo.em)
> example(haplo.em)

2.1 Example Data

The haplo.stats package contains three example data sets. The primary data set used in this manual is named hla.demo, which contains 11 loci from the HLA region on chromosome 6, with covariates, qualitative, and quantitative responses. Within /haplo.stats/data/hla.demo.tab the data is stored in tab-delimited format. Typically data stored in this format can be read in using read.table(). Since the data is provided in the package, we load the data in R using data() and view the names of the columns. Then to make the columns of hla.demo accessible without typing it each time, we attach it to the current session.

> # load and preview demo dataset stored in ~/haplo.stats/data/hla.demo.tab
> data(hla.demo)
> names(hla.demo)

[1] "resp"   "resp.cat"  "male"   "age"    "DPB.a1"  "DPB.a2"
[7] "DPA.a1"  "DPA.a2"   "DMA.a1"  "DMA.a2"  "DMB.a1"  "DMB.a2"
[13] "TAP1.a1" "TAP1.a2"  "TAP2.a1" "TAP2.a2" "DQB.a1"  "DQB.a2"
[19] "DQA.a1"  "DQA.a2"  "DRB.a1"  "DRB.a2"  "B.a1"    "B.a2"
[25] "A.a1"    "A.a2"

> # attach hla.demo to make columns available in the session
> attach(hla.demo)

The column names of hla.demo are shown above. They are defined as follows:

- **resp**: quantitative antibody response to measles vaccination
- **resp.cat**: a factor with levels "low", "normal", "high", for categorical antibody response
- **male**: gender code with 1="male", 0="female"
- **age**: age (in months) at immunization

The remaining columns are genotypes for 11 HLA loci, with a prefix name (e.g., "DQB") and a suffix for each of two alleles (".a1" and ".a2"). The variables in hla.demo can be accessed by typing hla.demo$ before their names, such as hla.demo$resp. Alternatively, it is easier for these examples to attach hla.demo, (as shown above with attach()) so the variables can be accessed by simply typing their names.
2.2 Creating a Genotype Matrix

Many of the functions require a matrix of genotypes, denoted below as geno. This matrix is arranged such that each locus has a pair of adjacent columns of alleles, and the order of columns corresponds to the order of loci on a chromosome. If there are K loci, then the number of columns of geno is $2^K$. Rows represent the alleles for each subject. For example, if there are three loci, in the order A-B-C, then the 6 columns of geno would be arranged as A.a1, A.a2, B.a1, B.a2, C.a1, C.a2. For illustration, three of the loci in hla.demo will be used to demonstrate some of the functions. Create a separate data frame for 3 of the loci, and call this geno. Then create a vector of labels for the loci.

```r
> geno <- hla.demo[,c(17,18,21:24)]
> label <- c("DQB","DRB","B")
```

The hla.demo data already had alleles in two columns for each locus. For many SNP datasets, the data is in a one column format, giving the count of the minor allele. To assist in converting this format to two columns, a function named geno1to2 has been added to the package. See its help file for more details.

2.3 Preview Missing Data: summaryGeno

Before performing a haplotype analysis, the user will want to assess missing genotype data to determine the completeness of the data. If many genotypes are missing, the functions may take a long time to compute results, or even run out of memory. For these reasons, the user may want to remove some of the subjects with a lot of missing data. This step can be guided by using the summaryGeno function, which checks for missing allele information and counts the number of potential haplotype pairs that are consistent with the observed data (see the Appendix for a description of this counting scheme).

The codes for missing alleles are defined by the parameter miss.val, a vector to define all possible missing value codes. Below, the result is saved in geno.desc, which is a data frame, so individual rows may be printed. Here we show the results for subjects 1-10, 80-85, and 135-140, some of which have missing alleles.

```r
> geno.desc <- summaryGeno(geno, miss.val=c(0,NA))
> print(geno.desc[c(1:10,80:85,135:140),])
```

<table>
<thead>
<tr>
<th></th>
<th>missing0</th>
<th>missing1</th>
<th>missing2</th>
<th>N.enum.rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
The columns with 'loc miss-' illustrate the number of loci missing either 0, 1, or 2 alleles, and the last column, num_enum_rows, illustrates the number of haplotype pairs that are consistent with the observed data. In the example above, subjects indexed by rows 81 and 137 have missing alleles. Subject #81 has one locus missing two alleles, while subject #137 has two loci missing two alleles. As indicated by num_enum_rows, subject #81 has 1,800 potential haplotype pairs, while subject #137 has nearly 130,000.

The 130,000 haplotype pairs is considered a large number, but haplo.em, haplo.score, and haplo.glm complete in roughly 3-6 minutes (depending on system limits or control parameter settings). If person #137 were removed, the methods would take less than half that time. It is preferred to keep people if they provide information to the analysis, given that run time and memory usage are not too much of a burden.

When a person has no genotype information, they do not provide information to any of the methods in haplo.stats. Furthermore, they cause a much longer run time. Below, using the table function on the third column of geno.desc, we can tabulate how many people are missing two alleles at any at of the three loci. If there were people missing two alleles at all three loci, they should be removed. The second command below shows how to make an index of which people to remove from hla.demo because they are missing all their alleles.

> # find if there are any people missing all alleles
> table(geno.desc[,3])

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>218</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

> ## create an index of people missing all alleles
> miss.all <- which(geno.desc[,3]==3)
> # use index to subset hla.demo
> hla.demo.updated <- hla.demo[-miss.all,]

2.4 Random Numbers and Setting Seed

Random numbers are used in several of the functions (e.g., to determine random starting frequencies within haplo.em, and to compute permutation p-values in haplo.score). To reproduce calculations involving random numbers, we set the seed values stored in a vector called .Random.seed. This
vector can be set using `set.seed()` before any function which uses random numbers. Section 6.7.2 shows one example of setting the seed for `haplo.glm`. We illustrate setting the seed below.

```r
> # this is how to set the seed for reproducing results where haplo.em is
> # involved, and also if simulations are run. In practice, don’t reset seed.
> seed <- c(17, 53, 1, 40, 37, 0, 62, 56, 5, 52, 12, 1)
> set.seed(seed)
```

3 Haplotype Frequency Estimation: `haplo.em`

3.1 Algorithm

For genetic markers measured on unrelated subjects, with linkage phase unknown, `haplo.em` computes maximum likelihood estimates of haplotype probabilities. Because there may be more than one pair of haplotypes that are consistent with the observed marker phenotypes, posterior probabilities of haplotype pairs for each subject are also computed. Unlike the usual EM which attempts to enumerate all possible haplotype pairs before iterating over the EM steps, our `progressive insertion` algorithm progressively inserts batches of loci into haplotypes of growing lengths, runs the EM steps, trims off pairs of haplotypes per subject when the posterior probability of the pair is below a specified threshold, and then continues these insertion, EM, and trimming steps until all loci are inserted into the haplotype. The user can choose the batch size. If the batch size is chosen to be all loci, and the threshold for trimming is set to 0, then this reduces to the usual EM algorithm. The basis of this `progressive insertion` algorithm is from the ”snphap” software by David Clayton[1]. Although some of the features and control parameters of `haplo.em` are modeled after `snphap`, there are substantial differences, such as extension to allow for more than two alleles per locus, and some other nuances on how the algorithm is implemented.

3.2 Example Usage

We use `haplo.em` on `geno` for the 3 loci defined above and save the result in an object named `save.em`, which has the `haplo.em` class. The print method would normally print all 178 haplotypes from `save.em`, but to keep the results short for this manual, we give a quick glance of the output by using the option `nlines=10`, which prints only the first 10 haplotypes of the full results. The `nlines` parameter has been employed in some of Haplo Stats’ print methods for when there are many haplotypes. In practice, it is best to exclude this parameter so that the default will print all results.

```r
> save.em <- haplo.em(geno=geno, locus.label=label, miss.val=c(0,NA))
> names(save.em)
[1] "lnlike"     ""         "lr"        "df.lr"
[5] "hap.prob"   "hap.prob.noLD" "converge"  "locus.label"
[9] "indx.subj"  "subj.id"   "post"       "hap1code"
[13] "hap2code"   "haplotype" "nreps"      "rows.rem"
[17] "max.pairs"  "control"
> print(save.em, nlines=10)
```
Haplotypes

<table>
<thead>
<tr>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>hap.freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>3</td>
<td>35</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>3</td>
<td>44</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>3</td>
<td>45</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>3</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>3</td>
<td>57</td>
</tr>
</tbody>
</table>

Details

lnlike = -1847.675
lr stat for no LD = 632.5085 , df = 125 , p-val = 0

Explanation of Results

The print methods shows the haplotypes and their estimated frequencies, followed by the final log-likelihood statistic and the lr stat for no LD, which is the likelihood ratio test statistic contrasting the lnlike for the estimated haplotype frequencies versus the lnlike under the null assuming that alleles from all loci are in linkage equilibrium. We note that the trimming by the progressive insertion algorithm can invalidate the lr stat and the degrees of freedom (df).

3.3 Summary Method

The summary method for a haplo.em object on save.em shows the list of haplotypes per subject, and their posterior probabilities:

> summary(save.em, nlines=7)

<table>
<thead>
<tr>
<th>subj.id</th>
<th>hap1code</th>
<th>hap2code</th>
<th>posterior</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>58</td>
<td>78</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>13</td>
<td>143</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>17</td>
<td>138</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>25</td>
<td>168</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>13</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>17</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>55</td>
<td>94</td>
</tr>
</tbody>
</table>
Explanation of Results

The first part of the *summary* output lists the subject id (row number of input *geno* matrix), the codes for the haplotypes of each pair, and the posterior probabilities of the haplotype pairs. The second part gives a table of the maximum number of pairs of haplotypes per subject, versus the number of pairs used in the final posterior probabilities. The haplotype codes remove the clutter of illustrating all the alleles of the haplotypes, but may not be as informative as the actual haplotypes themselves. To see the actual haplotypes, use the `show.haplo=TRUE` option, as in the following example.

```r
> # show full haplotypes, instead of codes
> summary(save.em, show.haplo=TRUE, nlines=7)
```

---

Subjects: Haplotype Codes and Posterior Probabilities

---

<table>
<thead>
<tr>
<th>subj.id</th>
<th>hap1.DQB</th>
<th>hap1.DRB</th>
<th>hap1.B</th>
<th>hap2.DQB</th>
<th>hap2.DRB</th>
<th>hap2.B</th>
<th>posterior</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>1</td>
<td>31</td>
<td>11</td>
<td>61</td>
<td>32</td>
<td>4</td>
<td>62</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>21</td>
<td>7</td>
<td>7</td>
<td>62</td>
<td>2</td>
<td>44</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>62</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>25</td>
<td>3</td>
<td>31</td>
<td>1</td>
<td>27</td>
<td>63</td>
<td>13</td>
<td>62</td>
</tr>
<tr>
<td>13.1</td>
<td>4</td>
<td>21</td>
<td>7</td>
<td>27</td>
<td>63</td>
<td>13</td>
<td>62</td>
</tr>
<tr>
<td>17.1</td>
<td>4</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>31</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>55</td>
<td>5</td>
<td>31</td>
<td>11</td>
<td>51</td>
<td>42</td>
<td>8</td>
<td>55</td>
</tr>
</tbody>
</table>

---

Number of haplotype pairs: max vs used

---

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>84</th>
<th>135</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>116</td>
<td>29</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1800</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>129600</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
3.4 Control Parameters for haplo.em

A set of control parameters can be passed together to haplo.em as the “control” argument. This is a list of parameters that control the EM algorithm based on progressive insertion of loci. The default values are set by a function called haplo.em.control (see help(haplo.em.control) for a complete description). Although the user can accept the default values, there are times when control parameters may need to be adjusted.

These parameters are defined below:

- **insert.batch.size**: Number of loci to be inserted in a single batch.

- **min.posterior**: Minimum posterior probability of haplotype pair, conditional on observed marker genotypes. Posteriors below this minimum value will have their pair of haplotypes "trimmed" off the list of possible pairs.

- **max.iter**: Maximum number of iterations allowed for the EM algorithm before it stops and prints an error.

- **n.try**: Number of times to try to maximize the lnlike by the EM algorithm. The first try will use, as initial starting values for the posteriors, either equal values or uniform random variables, as determined by random.start. All subsequent tries will use uniform random values as initial starting values for the posterior probabilities.

- **max.haps.limit**: Maximum number of haplotypes for the input genotypes. Within haplo.em, the first step is to try to allocate the sum of the result of geno.count.pairs(), if that exceeds max.haps.limit, start by allocating max.haps.limit. If that is exceeded in the progressive-insertions steps, the C function doubles the memory until it can longer request more.

One reason to adjust control parameters is for finding the global maximum of the log-likelihood. It can be difficult in particular for small sample sizes and many possible haplotypes. Different maximizations of the log-likelihood may result in different results from haplo.em, haplo.score, or haplo.glm when rerunning the analyses. The algorithm uses multiple attempts to maximize the log-likelihood, starting each attempt with random starting values. To increase the chance of finding the global maximum of the log-likelihood, the user can increase the number of attempts (n.try), increase the batch size (insert.batch.size), or decrease the trimming threshold for posterior probabilities (min.posterior).

Another reason to adjust control parameters is when the algorithm runs out of memory because there are too many haplotypes. If max.haps.limit is exceeded when a batch of markers is added, the algorithm requests twice as much memory until it runs out. One option is to set max.haps.limit to a different value, either to make haplo.em request more memory initially, or to request more memory in smaller chunks. Another solution is to make the algorithm trim the number of haplotypes more aggressively by decreasing insert.batch.size or increasing min.posterior. Any changes to these parameters should be made with caution, and not drastically different from the default values. For instance, the default for min.posterior used to be $1e^{-7}$, and in some rare circumstances with many markers in only moderate linkage disequilibrium, some subjects had all their possible haplotype pairs trimmed. The default is now set at $1e^{-9}$, and we recommend not increasing min.posterior much greater than $1e^{-7}$.

The example below gives the command for increasing the number of tries to 20, and the batch size to 2, since not much more can be done for three markers.
3.5 Haplotype Frequencies by Group Subsets

To compute the haplotype frequencies for each level of a grouping variable, use the function `haplo.group`. The following example illustrates the use of a binomial response based on `resp.cat`, `y.bin`, that splits the subjects into two groups.

```r
> ## run haplo.em on sub-groups
> ## create ordinal and binary variables
> y.bin <- 1*(resp.cat=="low")
> group.bin <- haplo.group(y.bin, geno, locus.label=label, miss.val=0)
> print(group.bin, nlines=15)

Counts per Grouping Variable Value
--------------------------------------------------------------------------------
group
0 1
157 63
--------------------------------------------------------------------------------
Haplotype Frequencies By Group
--------------------------------------------------------------------------------
   DQB DRB B Total y.bin.0 y.bin.1
 1  21  1  8 0.00232 0.00335 NA
 2  21 10  8 0.00181 0.00318 NA
 3  21 13  8 0.00274   NA   NA
 4  21  2 18 0.00227 0.00318 NA
 5  21  2  7 0.00227 0.00318 NA
 6  21  3 18 0.00229 0.00637 NA
 7  21  3 35 0.00570 0.00639 NA
 8  21  3 44 0.00378 0.00333 0.01587
 9  21  3 45 0.00227   NA   NA
10 21  3 49 0.00227   NA   NA
11 21  3 57 0.00227   NA   NA
12 21  3 70 0.00227   NA 0.00000
13 21  3  8 0.10408 0.06974 0.19048
14 21  4 62 0.00455 0.00637 NA
15 21  7 13 0.01072   NA 0.02381
```

Explanation of Results

The `group.bin` object can be very large, depending on the number of possible haplotypes, so only
a portion of the output is illustrated above (limited again by nlines). The first section gives a short summary of how many subjects appear in each of the groups. The second section is a table with the following columns:

- The first column gives row numbers.
- The next columns (3 in this example) illustrate the alleles of the haplotypes.
- Total are the estimated haplotype frequencies for the entire data set.
- The last columns are the estimated haplotype frequencies for the subjects in the levels of the group variable (y.bin=0 and y.bin=1 in this example). Note that some haplotype frequencies have an NA, which appears when the haplotypes do not occur in the subgroups.

4 Power and Sample Size for Haplotype Association Studies

It is known that using haplotypes has greater power than single-markers to detect genetic association in some circumstances. There is little guidance, however, in determining sample size and power under different circumstances, some of which include: marker type, dominance, and effect size. The haplo.stats package now includes functions to calculate sample size and power for haplotype association studies, which is flexible to handle these multiple circumstances.

Based on work in Schaid 2005[2], we can take a set of haplotypes with their population frequencies, assign a risk to a subset of the haplotypes, then determine either the sample size to achieve a stated power, or the power for a stated sample size. Sample size and power can be calculated for either quantitative traits or case-control studies.

4.1 Quantitative Traits: haplo.power.qt

We assume that quantitative traits will be modeled by a linear regression. Some well-known tests for association between haplotypes and the trait include score statistics[3] and an F-test[4]. For both types of tests, power depends on the amount of variance in the trait that is explained by haplotypes, or a multiple correlation coefficient, $R^2$. Rather than specifying the haplotype coefficients directly, we calculate the vector of coefficients based on an $R^2$ value.

In the example below, we load an example set of haplotypes that contain 5 markers, and specify the indices of the at-risk haplotypes; in this case, whichever haplotype has allele 1 at the 2nd and 3rd markers. We set the first haplotype (most common) as the baseline. With these values we calculate the vector of coefficients for haplotype effects from find.haplo.beta.qt using an $R^2 = 0.01$. Next, we use haplo.power.qt to calculate the sample size for the set of haplotypes and their coefficients, type-I error (alpha) set to 0.05, power at 80%, and the same mean and variance used to get haplotype coefficients. Then we use the sample size needed for 80% power for un-phased haplotypes (2,826) to get the power for both phased and un-phased haplotypes.

```r
> # load a set of haplotypes (hap-1 from Schaid 2005)
> data(hapPower.demo)
> #### an example using save.em hla markers may go like this.
> # keep <- which(save.em$hap.prob > .004)  # get an index of non-rare haps
```

14
> # hfreq <- save.em$hap.prob[keep]
> # hmat <- save.em$haplotype[keep]
> # hrisk <- which(hmat[,1]==31 & hmat[,2]==11)  # contains 3 haps with freq=.01
> # hbase <- 4  # 4th hap has mas freq of .103
> ###
> # separate the haplotype matrix and the frequencies
> hmat <- hapPower.demo[,,-6]
> hfreq <- hapPower.demo[,6]
> ## Define risk haplotypes as those with "1" allele at loc2 and loc3
> hrisk <- which(hmat$loc.2==1 & hmat$loc.3==1)
> # define index for baseline haplotype
> hbase <- 1
> hbeta.list <- find.haplo.beta.qt(haplo=hmat, haplo.freq=hfreq, base.index=hbase,
>                                   haplo.risk=hrisk, r2=.01, y.mu=0, y.var=1)
> hbeta.list
$\{r2
[1] 0.01
$
$beta
[1,] -0.038924970 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
[7,] 0.000000000 0.276367310 0.000000000 0.276367310 0.000000000 0.276367310 0.000000000 0.276367310 0.276367310 0.000000000 0.276367310 0.276367310 0.000000000 0.276367310 0.276367310 0.000000000 0.000000000 0.276367310 0.276367310
[13,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.276367310 0.276367310 0.276367310 0.276367310 0.000000000 0.276367310 0.276367310 0.000000000 0.276367310 0.276367310 0.000000000 0.276367310 0.276367310 0.000000000
[19,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
$
$base.index
[1] 1
$
$haplo.risk
[1] 8 10 16 17

> ss.qt <- haplo.power.qt(hmat, hfreq, hbase, hbeta.list$beta,
>                         y.mu=0, y.var=1, alpha=.05, power=.80)
> ss.qt
$ss.phased.haplo
[1] 2091

$ss.unphased.haplo
[1] 2826

$power.phased.haplo
[1] 0.8

$power.unphased.haplo
[1] 0.8

15
> power.qt <- haplo.power.qt(hmat, hfreq, hbase, hbeta.list$beta,
+                         y.mu=0, y.var=1, alpha=.05, sample.size=2826)
> power.qt

$ss.phased.haplo
[1] 2826

$ss.unphased.haplo
[1] 2826

$power.phased.haplo
[1] 0.9282451

$power.unphased.haplo
[1] 0.8000592

4.2 Case-Control Studies: haplo.power.cc

The steps to compute sample size and power for case-control studies is similar to the steps for quantitative traits. If we assume a log-additive model for haplotype effects, the haplotype coefficients can be specified first as odds ratios (OR), and then converted to logistic regression coefficients according to \( \log(OR) \).

In the example below, we assume the same baseline and risk haplotypes defined in section 4.1, give the risk haplotypes an odds ratio of 1.50, and specify a population disease prevalence of 10%. We also assume cases make up 50% (case.frac) of the study's subjects. We first compute the sample size for this scenario for Type-I error (alpha) at 0.05 and 80% power, and then compute power for the sample size required for un-phased haplotypes (4,566).

> ## get power and sample size for quantitative response
> ## get beta vector based on odds ratios
> cc.OR <- 1.5
> # determine beta regression coefficients for risk haplotypes
> hbeta.cc <- numeric(length(hfreq))
> hbeta.cc[hrisk] <- log(cc.OR)
> # Compute sample size for stated power
> ss.cc <- haplo.power.cc(hmat, hfreq, hbase, hbeta.cc, case.frac=.5, prevalence=.1,
+                        alpha=.05, power=.8)
> ss.cc

$ss.phased.haplo
[1] 3454

$ss.unphased.haplo
5 Haplotype Score Tests: *haplo.score*

The *haplo.score* routine is used to compute score statistics to test association between haplotypes and a wide variety of traits, including binary, ordinal, quantitative, and Poisson. This function provides several different global and haplotype-specific tests for association and allows for adjustment for non-genetic covariates. Haplotype effects can be specified as additive, dominant, or recessive. This method also has an option to compute permutation p-values, which may be needed for sparse data when distribution assumptions may not be met. Details on the background and theory of the score statistics can be found in Schaid et al.[3].

5.1 Quantitative Trait Analysis

First, we assess a haplotype association with a quantitative trait in *hla.demo* called *resp*. To tell *haplo.score* the trait is quantitative, specify the parameter *trait.type="gaussian"* (a reminder that a gaussian distribution is assumed for the error terms). The other arguments, all set to default values, are explained in the help file. Note that rare haplotypes can result in unstable variance estimates, and hence unreliable test statistics for rare haplotypes. We restrict the analysis to get scores for haplotypes with a minimum sample count using *min.count=5*. For more explanation on handling rare haplotypes, see section 5.6. Below is an example of running *haplo.score* with a
quantitative trait, then viewing the results using the print method for the haplo.score class. (again, output shortened by nlines).

```R
> # score statistics w/ Gaussian trait
> score.gaus.add <- haplo.score(resp, geno, trait.type="gaussian",
+                                 min.count=5,
+                                 locus.label=label, simulate=FALSE)
> print(score.gaus.add, nlines=10)
```

--------------------------------------------------------------------------------
Haplotype Effect Model: additive
--------------------------------------------------------------------------------
Global Score Statistics
--------------------------------------------------------------------------------
global-stat = 30.6353, df = 18, p-val = 0.03171
--------------------------------------------------------------------------------
Haplotype-specific Scores
--------------------------------------------------------------------------------

<table>
<thead>
<tr>
<th></th>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>Hap-Freq</th>
<th>Hap-Score</th>
<th>p-val</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>21</td>
<td>3</td>
<td>8</td>
<td>0.10408</td>
<td>-2.39631</td>
<td>0.01656</td>
</tr>
<tr>
<td>[2,]</td>
<td>31</td>
<td>4</td>
<td>44</td>
<td>0.02849</td>
<td>-2.24273</td>
<td>0.02491</td>
</tr>
<tr>
<td>[3,]</td>
<td>51</td>
<td>1</td>
<td>44</td>
<td>0.01731</td>
<td>-0.99357</td>
<td>0.32043</td>
</tr>
<tr>
<td>[4,]</td>
<td>63</td>
<td>13</td>
<td>44</td>
<td>0.01606</td>
<td>-0.84453</td>
<td>0.39837</td>
</tr>
<tr>
<td>[5,]</td>
<td>63</td>
<td>2</td>
<td>7</td>
<td>0.01333</td>
<td>-0.50736</td>
<td>0.6119</td>
</tr>
<tr>
<td>[6,]</td>
<td>32</td>
<td>4</td>
<td>60</td>
<td>0.0306</td>
<td>-0.46606</td>
<td>0.64118</td>
</tr>
<tr>
<td>[7,]</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>0.02332</td>
<td>-0.41942</td>
<td>0.67491</td>
</tr>
<tr>
<td>[8,]</td>
<td>62</td>
<td>2</td>
<td>44</td>
<td>0.01367</td>
<td>-0.26221</td>
<td>0.79316</td>
</tr>
<tr>
<td>[9,]</td>
<td>62</td>
<td>2</td>
<td>18</td>
<td>0.01545</td>
<td>-0.21493</td>
<td>0.82982</td>
</tr>
<tr>
<td>[10,]</td>
<td>51</td>
<td>1</td>
<td>27</td>
<td>0.01505</td>
<td>0.01539</td>
<td>0.98772</td>
</tr>
</tbody>
</table>

Explanation of Results

First, the model effect chosen by haplo.effect is printed across the top. The section Global Score Statistics shows results for testing an overall association between haplotypes and the response. The global-stat has an asymptotic $\chi^2$ distribution, with degrees of freedom (df) and p-value as indicated. Next, Haplotype-specific scores are given in a table format. The column descriptions are as follows:

- The first column gives row numbers.
- The next columns (3 in this example) illustrate the alleles of the haplotypes.
- $Hap$-Freq is the estimated frequency of the haplotype in the pool of all subjects.
- $Hap$-Score is the score for the haplotype, the results are sorted by this value. Note, the score statistic should not be interpreted as a measure of the haplotype effect.
• p-val is the asymptotic \( \chi^2 \) p-value, calculated from the square of the score statistic.

### 5.2 Binary Trait Analysis

Let us assume that "low" responders are of primary interest, so we create a binary trait that has values of 1 when \( \text{resp.cat} \) is "low", and 0 otherwise. Then in \textit{haplo.score} specify the parameter \textit{trait.type}="binomial".

```r
> # scores, binary trait
> y.bin <- 1*(resp.cat=='low')
> score.bin <- haplo.score(y.bin, geno, trait.type="binomial",
+ x.adj = NA, min.count=5,
+ haplo.effect="additive", locus.label=label,
+ miss.val=0, simulate=FALSE)
> print(score.bin, nlines=10)
```

<table>
<thead>
<tr>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>Hap-Freq</th>
<th>Hap-Score</th>
<th>p-val</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>62</td>
<td>2</td>
<td>7</td>
<td>0.05098</td>
<td>-2.19387</td>
</tr>
<tr>
<td>[2,]</td>
<td>51</td>
<td>1</td>
<td>35</td>
<td>0.03018</td>
<td>-1.58421</td>
</tr>
<tr>
<td>[3,]</td>
<td>63</td>
<td>13</td>
<td>7</td>
<td>0.01655</td>
<td>-1.56008</td>
</tr>
<tr>
<td>[4,]</td>
<td>21</td>
<td>7</td>
<td>7</td>
<td>0.01246</td>
<td>-1.47495</td>
</tr>
<tr>
<td>[5,]</td>
<td>32</td>
<td>4</td>
<td>7</td>
<td>0.01678</td>
<td>-1.00091</td>
</tr>
<tr>
<td>[6,]</td>
<td>32</td>
<td>4</td>
<td>62</td>
<td>0.02349</td>
<td>-0.6799</td>
</tr>
<tr>
<td>[7,]</td>
<td>51</td>
<td>1</td>
<td>27</td>
<td>0.01505</td>
<td>-0.66509</td>
</tr>
<tr>
<td>[8,]</td>
<td>31</td>
<td>11</td>
<td>35</td>
<td>0.01754</td>
<td>-0.5838</td>
</tr>
<tr>
<td>[9,]</td>
<td>31</td>
<td>11</td>
<td>51</td>
<td>0.01137</td>
<td>-0.43721</td>
</tr>
<tr>
<td>[10,]</td>
<td>51</td>
<td>1</td>
<td>44</td>
<td>0.01731</td>
<td>0.00826</td>
</tr>
</tbody>
</table>

### 5.3 Ordinal Trait Analysis

To create an ordinal trait, here we convert \( \text{resp.cat} \) (described above) to numeric values, \( y.ord \) (with levels 1, 2, 3). For \textit{haplo.score}, use \( y.ord \) as the response variable, and set the parameter \textit{trait.type} = "ordinal".
> # scores w/ ordinal trait
> y.ord <- as.numeric(resp.cat)
> score.ord <- haplo.score(y.ord, geno, trait.type="ordinal",
>                         x.adj = NA, min.count=5,
>                         locus.label=label,
>                         miss.val=0, simulate=FALSE)
> print(score.ord, nlines=7)

--------------------------------------------------------------------------------
Haplotype Effect Model: additive
--------------------------------------------------------------------------------
Global Score Statistics
--------------------------------------------------------------------------------
global-stat = 15.23209, df = 18, p-val = 0.64597
--------------------------------------------------------------------------------
Haplotype-specific Scores
--------------------------------------------------------------------------------

<table>
<thead>
<tr>
<th></th>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>Hap-Freq</th>
<th>Hap-Score</th>
<th>p-val</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>32</td>
<td>4</td>
<td>62</td>
<td>0.02349</td>
<td>-2.17133</td>
<td>0.02991</td>
</tr>
<tr>
<td>[2,]</td>
<td>21</td>
<td>3</td>
<td>8</td>
<td>0.10408</td>
<td>-1.34661</td>
<td>0.17811</td>
</tr>
<tr>
<td>[3,]</td>
<td>32</td>
<td>4</td>
<td>7</td>
<td>0.01678</td>
<td>-1.09487</td>
<td>0.27357</td>
</tr>
<tr>
<td>[4,]</td>
<td>62</td>
<td>2</td>
<td>7</td>
<td>0.05098</td>
<td>-0.96874</td>
<td>0.33268</td>
</tr>
<tr>
<td>[5,]</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>0.02332</td>
<td>-0.83747</td>
<td>0.40233</td>
</tr>
<tr>
<td>[6,]</td>
<td>63</td>
<td>13</td>
<td>7</td>
<td>0.01655</td>
<td>-0.80787</td>
<td>0.41917</td>
</tr>
<tr>
<td>[7,]</td>
<td>21</td>
<td>7</td>
<td>7</td>
<td>0.01246</td>
<td>-0.63316</td>
<td>0.52663</td>
</tr>
</tbody>
</table>

Warning for Ordinal Traits

When analyzing an ordinal trait with adjustment for covariates (using the x.adj option), the software requires the rms package, distributed by Frank Harrell [5]. If the user does not have these packages installed, then it will not be possible to use the x.adj option. However, the unadjusted scores for an ordinal trait (using the default option x.adj=NA) do not require these packages. Check the list of your local packages in the list shown from entering library() in your prompt.

5.4 Haplotype Scores, Adjusted for Covariates

To adjust for covariates in haplo.score, first set up a matrix of covariates from the example data. For example, use a column for male (1 if male; 0 if female), and a second column for age. Then pass the matrix to haplo.score using parameter x.adj. The results change slightly in this example.

> # score w/gaussian, adjusted by covariates
> x.ma <- cbind(male, age)
> score.gaus.adj <- haplo.score(resp, geno, trait.type="gaussian",
+ x.adj = x.ma, min.count=5,
+ locus.label=label, simulate=FALSE)
> print(score.gaus.adj, nlines=10)

---
Haplotype Effect Model: additive
---

Global Score Statistics
---
global-stat = 31.02908, df = 18, p-val = 0.02857
---

Haplotype-specific Scores
---

<table>
<thead>
<tr>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>Hap-Freq</th>
<th>Hap-Score</th>
<th>p-val</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>21</td>
<td>3</td>
<td>8</td>
<td>0.10408</td>
<td>-2.4097</td>
</tr>
<tr>
<td>[2,]</td>
<td>31</td>
<td>4</td>
<td>44</td>
<td>0.02849</td>
<td>-2.25293</td>
</tr>
<tr>
<td>[3,]</td>
<td>51</td>
<td>1</td>
<td>44</td>
<td>0.01731</td>
<td>-0.98763</td>
</tr>
<tr>
<td>[4,]</td>
<td>63</td>
<td>13</td>
<td>44</td>
<td>0.01606</td>
<td>-0.83952</td>
</tr>
<tr>
<td>[5,]</td>
<td>63</td>
<td>2</td>
<td>7</td>
<td>0.01333</td>
<td>-0.48483</td>
</tr>
<tr>
<td>[6,]</td>
<td>32</td>
<td>4</td>
<td>60</td>
<td>0.0306</td>
<td>-0.46476</td>
</tr>
<tr>
<td>[7,]</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>0.02332</td>
<td>-0.41249</td>
</tr>
<tr>
<td>[8,]</td>
<td>62</td>
<td>2</td>
<td>44</td>
<td>0.01367</td>
<td>-0.26443</td>
</tr>
<tr>
<td>[9,]</td>
<td>62</td>
<td>2</td>
<td>18</td>
<td>0.01545</td>
<td>-0.20425</td>
</tr>
<tr>
<td>[10,]</td>
<td>51</td>
<td>1</td>
<td>27</td>
<td>0.01505</td>
<td>0.02243</td>
</tr>
</tbody>
</table>

5.5 Plots and Haplotype Labels

A convenient way to view results from \textit{haplo.score} is a plot of the haplotype frequencies (\textit{Hap-Freq}) versus the haplotype score statistics (\textit{Hap-Score}). This plot, and the syntax for creating it, are shown in Figure 1.

Some points on the plot may be of interest. To identify individual points on the plot, use \textit{locator.haplo(score.gaus)}, which is similar to \textit{locator()}. Use the mouse to select points on the plot. After points are chosen, click on the middle mouse button, and the points are labeled with their haplotype labels. Note, in constructing Figure 1, we had to define which points to label, and then assign labels in the same way as done within the \textit{locator.haplo} function.
> ## plot score vs. frequency, gaussian response
> plot(score.gaus.add, pch="o")
> ## locate and label pts with their haplotypes
> ## works similar to locator() function
> #> pts.haplo <- locator.haplo(score.gaus)
> pts.haplo <- list(x.coord=c(0.05098, 0.03018, .100),
+                  y.coord=c(2.1582, 0.45725, -2.1566),
> text(x=pts.haplo$x.coord, y=pts.haplo$y.coord, labels=pts.haplo$hap.txt)

Figure 1: Haplotype Statistics: Score vs. Frequency, Quantitative Response
5.6 Skipping Rare Haplotypes

For the \texttt{haplo.score}, the \texttt{skip.haplo} and \texttt{min.count} parameters control which rare haplotypes are pooled into a common group. The \texttt{min.count} parameter is a recent addition to \texttt{haplo.score}, yet it does the same task as \texttt{skip.haplo} and is the same idea as \texttt{haplo.min.count} used in \texttt{haplo.glm.control} for \texttt{haplo.glm}. As a guideline, you may wish to set \texttt{min.count} to calculate scores for haplotypes with expected haplotype counts of 5 or greater in the sample. We concentrate on this expected count because it adjusts to the size of the input data. If \(N\) is the number of subjects and \(f\) the haplotype frequency, then the expected haplotype count is \(\text{count} = 2 \times N \times f\). Alternatively, you can choose \(\text{skip.haplo} = \frac{\text{count}}{2 \times N}\).

In the following example we try a different cut-off than before, \texttt{min.count=10}, which corresponds to \texttt{skip.haplo} of \(10 \div (2 \times 220) = .045\). In the output, see that the global statistic, degrees of freedom, and p-value change because of the fewer haplotypes, while the haplotype-specific scores do not change.

```r
> # increase skip.haplo, expected hap counts = 10
> score.gaus.min10 <- haplo.score(resp, geno, trait.type="gaussian",
+ x.adj = NA, min.count=10,
+ locus.label=label, simulate=FALSE)
> print(score.gaus.min10)
```

<table>
<thead>
<tr>
<th>Haplotype Effect Model: additive</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>Global Score Statistics</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>global-stat = 20.66451, df = 7, p-val = 0.0043</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Haplotype-specific Scores</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DQB DRB B</th>
<th>Hap-Freq</th>
<th>Hap-Score</th>
<th>p-val</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,] 21 3 8</td>
<td>0.10408</td>
<td>-2.39631</td>
<td>0.01656</td>
</tr>
<tr>
<td>[2,] 31 4 44</td>
<td>0.02849</td>
<td>-2.24273</td>
<td>0.02491</td>
</tr>
<tr>
<td>[3,] 32 4 60</td>
<td>0.0306</td>
<td>-0.46606</td>
<td>0.64118</td>
</tr>
<tr>
<td>[4,] 21 7 44</td>
<td>0.02332</td>
<td>-0.41942</td>
<td>0.67491</td>
</tr>
<tr>
<td>[5,] 51 1 35</td>
<td>0.03018</td>
<td>0.69696</td>
<td>0.48583</td>
</tr>
<tr>
<td>[6,] 32 4 62</td>
<td>0.02349</td>
<td>2.37619</td>
<td>0.01749</td>
</tr>
<tr>
<td>[7,] 62 2 7</td>
<td>0.05098</td>
<td>2.39795</td>
<td>0.01649</td>
</tr>
</tbody>
</table>

5.7 Score Statistic Dependencies: the \texttt{eps.svd} parameter

The global score test is calculated using the vector of scores and the generalized inverse of their variance-covariance matrix, performed by the \texttt{Ginv} function. This function determines the rank.
of the variance matrix by its singular value decomposition, and an epsilon value is used as the cut-off for small singular values. If all of the haplotypes in the sample are scored, then there is dependence between them and the variance matrix is not of full rank. However, it is more often the case that one or more rare haplotypes are not scored because of low frequency. It is not clear how strong the dependencies are between the remaining score statistics, and likewise, there is disparity in calculating the rank of the variance matrix. For these instances we give the user control over the epsilon parameter for `haplo.score` with `eps.svd`.

We have seen instances where the global score test had a very significant p-value, but none of the haplotype-specific scores showed strong association. In such instances, we found the default epsilon value in `Ginv` was incorrectly considering the variance matrix as having full rank, and the misleading global score test was corrected when we increased epsilon for `Ginv`. We now set the default for `eps.svd` at $1e^{-5}$, which seems to perform better in the troublesome circumstances than the previous default of $1e^{-6}$.

### 5.8 Haplotype Model Effect

A recent addition to `haplo.score` is the ability to select non-additive effects to score haplotypes. The possible effects for haplotypes are additive, dominant, and recessive. Under recessive effects, fewer haplotypes may be scored, because subjects are required to be homozygous for haplotypes. Furthermore, there would have to be `min.count` such persons in the sample to have the recessive effect scored. Therefore, a recessive model should only be used on samples with common haplotypes. In the example below with the gaussian response, set the haplotype effect to dominant using parameter `haplo_effect = "dominant"`. Notice the results change slightly compared to the `score.gaus.add` results above.

```r
> # score w/gaussian, dominant effect
> score.gaus.dom <- haplo.score(resp, geno, trait.type="gaussian",
+ x.adj=NA, min.count=5,
+ haplo.effect="dominant", locus.label=label, sim=FALSE)
> print(score.gaus.dom, nlines=10)
```

```
Haplotype Effect Model: dominant

Global Score Statistics

global-stat = 29.56133, df = 18, p-val = 0.04194
```

```r
```
5.9 Simulation p-values

When `simulate=TRUE`, `haplo.score` gives simulated p-values. Simulated haplotype score statistics are the re-calculated score statistics from a permuted re-ordering of the trait and covariates and the original ordering of the genotype matrix. The simulated p-value for the global score statistic (Global sim. p-val) is the number of times the simulated global score statistic exceeds the observed, divided by the total number of simulations. Likewise, simulated p-value for the maximum score statistic (Max-stat sim. p-val) is the number of times the simulated maximum haplotype score statistic exceeds the observed maximum score statistic, divided by the total number of simulations. The maximum score statistic is the maximum of the square of the haplotype-specific score statistics, which has an unknown distribution, so its significance can only be given by the simulated p-value. Intuitively, if only one or two haplotypes are associated with the trait, the maximum score statistic should have greater power to detect association than the global statistic.

The `score.sim.control` function manages control parameters for simulations. The `haplo.score` function employs the simulation p-value precision criteria of Besag and Clifford[6]. These criteria ensure that the simulated p-values for both the global and the maximum score statistics are precise for small p-values. The algorithm performs a user-defined minimum number of permutations (min.sim) to guarantee sufficient precision for the simulated p-values for score statistics of individual haplotypes. Permutations beyond this minimum are then conducted until the sample standard errors for simulated p-values for both the global-stat and max-stat score statistics are less than a threshold (p.threshold * p-value). The default value for p.threshold=1/2 provides a two-sided 95% confidence interval for the p-value with a width that is approximately as wide as the p-value itself. Effectively, simulations are more precise for smaller p-values. The following example illustrates computation of simulation p-values with min.sim=1000.

```r
> # simulations when binary response
> score.bin.sim <- haplo.score(y.bin, geno, trait.type="binomial",
+ x.adj = NA, locus.label=label, min.count=5,
+ simulate=TRUE, sim.control = score.sim.control() )
> print(score.bin.sim)

Haplotype Effect Model: additive

5.9 Simulation p-values

When simulate = TRUE, haplo.score gives simulated p-values. Simulated haplotype score statistics are the re-calculated score statistics from a permuted re-ordering of the trait and covariates and the original ordering of the genotype matrix. The simulated p-value for the global score statistic (Global sim. p-val) is the number of times the simulated global score statistic exceeds the observed, divided by the total number of simulations. Likewise, simulated p-value for the maximum score statistic (Max-stat sim. p-val) is the number of times the simulated maximum haplotype score statistic exceeds the observed maximum score statistic, divided by the total number of simulations. The maximum score statistic is the maximum of the square of the haplotype-specific score statistics, which has an unknown distribution, so its significance can only be given by the simulated p-value. Intuitively, if only one or two haplotypes are associated with the trait, the maximum score statistic should have greater power to detect association than the global statistic.

The score.sim.control function manages control parameters for simulations. The haplo.score function employs the simulation p-value precision criteria of Besag and Clifford[6]. These criteria ensure that the simulated p-values for both the global and the maximum score statistics are precise for small p-values. The algorithm performs a user-defined minimum number of permutations (min.sim) to guarantee sufficient precision for the simulated p-values for score statistics of individual haplotypes. Permutations beyond this minimum are then conducted until the sample standard errors for simulated p-values for both the global-stat and max-stat score statistics are less than a threshold (p.threshold * p-value). The default value for p.threshold = 1/2 provides a two-sided 95% confidence interval for the p-value with a width that is approximately as wide as the p-value itself. Effectively, simulations are more precise for smaller p-values. The following example illustrates computation of simulation p-values with min.sim=1000.

```
Global Score Statistics

global-stat = 33.70125, df = 18, p-val = 0.01371

Global Simulation p-value Results

Global sim. p-val = 0.0095
Max-Stat sim. p-val = 0.00563
Number of Simulations, Global: 2842 , Max-Stat: 2842

Haplotype-specific Scores

<table>
<thead>
<tr>
<th></th>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>Hap-Freq</th>
<th>Hap-Score</th>
<th>p-val</th>
<th>sim p-val</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>62</td>
<td>2</td>
<td>7</td>
<td>0.05098</td>
<td>-2.19387</td>
<td>0.02824</td>
<td>0.02991</td>
</tr>
<tr>
<td>[2,]</td>
<td>51</td>
<td>1</td>
<td>35</td>
<td>0.03018</td>
<td>-1.58421</td>
<td>0.11315</td>
<td>0.13863</td>
</tr>
<tr>
<td>[3,]</td>
<td>63</td>
<td>13</td>
<td>7</td>
<td>0.01655</td>
<td>-1.56008</td>
<td>0.11874</td>
<td>0.19177</td>
</tr>
<tr>
<td>[4,]</td>
<td>21</td>
<td>7</td>
<td>7</td>
<td>0.01246</td>
<td>-1.47495</td>
<td>0.14023</td>
<td>0.15588</td>
</tr>
<tr>
<td>[5,]</td>
<td>32</td>
<td>4</td>
<td>7</td>
<td>0.01678</td>
<td>-1.00091</td>
<td>0.31687</td>
<td>0.25123</td>
</tr>
<tr>
<td>[6,]</td>
<td>32</td>
<td>4</td>
<td>62</td>
<td>0.02349</td>
<td>-0.6799</td>
<td>0.49657</td>
<td>0.47467</td>
</tr>
<tr>
<td>[7,]</td>
<td>51</td>
<td>1</td>
<td>27</td>
<td>0.01505</td>
<td>-0.66509</td>
<td>0.50599</td>
<td>0.63089</td>
</tr>
<tr>
<td>[8,]</td>
<td>31</td>
<td>11</td>
<td>35</td>
<td>0.01754</td>
<td>-0.5838</td>
<td>0.55936</td>
<td>0.6506</td>
</tr>
<tr>
<td>[9,]</td>
<td>31</td>
<td>11</td>
<td>51</td>
<td>0.01137</td>
<td>-0.43721</td>
<td>0.66196</td>
<td>0.91872</td>
</tr>
<tr>
<td>[10,]</td>
<td>51</td>
<td>1</td>
<td>44</td>
<td>0.01731</td>
<td>0.00826</td>
<td>0.99341</td>
<td>1</td>
</tr>
<tr>
<td>[11,]</td>
<td>32</td>
<td>4</td>
<td>60</td>
<td>0.0306</td>
<td>0.03181</td>
<td>0.97462</td>
<td>0.95074</td>
</tr>
<tr>
<td>[12,]</td>
<td>62</td>
<td>2</td>
<td>44</td>
<td>0.01367</td>
<td>0.16582</td>
<td>0.8683</td>
<td>0.91872</td>
</tr>
<tr>
<td>[13,]</td>
<td>63</td>
<td>13</td>
<td>44</td>
<td>0.01606</td>
<td>0.22059</td>
<td>0.82541</td>
<td>0.7266</td>
</tr>
<tr>
<td>[14,]</td>
<td>63</td>
<td>2</td>
<td>7</td>
<td>0.01333</td>
<td>0.2982</td>
<td>0.76555</td>
<td>0.89163</td>
</tr>
<tr>
<td>[15,]</td>
<td>62</td>
<td>2</td>
<td>18</td>
<td>0.01545</td>
<td>0.78854</td>
<td>0.43038</td>
<td>0.6608</td>
</tr>
<tr>
<td>[16,]</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>0.02332</td>
<td>0.84562</td>
<td>0.39776</td>
<td>0.39796</td>
</tr>
<tr>
<td>[17,]</td>
<td>31</td>
<td>4</td>
<td>44</td>
<td>0.02849</td>
<td>2.50767</td>
<td>0.01215</td>
<td>0.01161</td>
</tr>
<tr>
<td>[18,]</td>
<td>21</td>
<td>3</td>
<td>8</td>
<td>0.10408</td>
<td>3.77763</td>
<td>0.00016</td>
<td>0.00035</td>
</tr>
</tbody>
</table>

6 Regression Models: haplo.glm

The haplo.glm function computes the regression of a trait on haplotypes, and possibly other covariates and their interactions with haplotypes. We currently support the gaussian, binomial, and Poisson families of traits with their canonical link functions. The effects of haplotypes on the link function can be modeled as either additive, dominant (heterozygotes and homozygotes for a particular haplotype assumed to have equivalent effects), or recessive (homozygotes of a particular
haplotype considered to have an alternative effect on the trait). The basis of the algorithm is a
two-step iteration process; the posterior probabilities of pairs of haplotypes per subject are used as
weights to update the regression coefficients, and the regression coefficients are used to update the
haplotype posterior probabilities. See Lake et al.[7] for details.

6.1 New and Updated Methods for haplo.glm

We initially wrote haplo.glm without much effort of making it work with R’s glm class methods.
We have now refined the haplo.glm class to look and act as much like a glm object as possible with
methods defined specifically for the haplo.glm class. We provide print and summary methods that
make use of the corresponding methods for glm and then add extra information for the haplotypes
and their frequencies. Furthermore, we have defined for the haplo.glm class some of the standard
methods for regression fits, including residuals, fitted.values, vcov, and anova. We describe the
challenges that haplotype regression presents with these methods in section 7.

6.2 Preparing the data.frame for haplo.glm

A critical distinction between haplo.glm and all other functions in Haplo Stats is that the definition
of the regression model follows the S/R formula standard (see lm or glm). So, a data.frame must
be defined, and this object must contain the trait and other optional covariates, plus a special kind
of genotype matrix (geno.glm for this example) that contains the genotypes of the marker loci. We
require the genotype matrix to be prepared using setupGeno(), which handles character, numeric,
or factor alleles, and keeps the columns of the genotype matrix as a single unit when inserting into
(and extracting from) a data.frame. The setupGeno function recodes all missing genotype value
codes given by miss.val to NA, and also recodes alleles to integer values. The original allele codes
are preserved within an attribute of geno.glm, and are utilized within haplo.glm. The returned
object has class model.matrix, and it can be included in a data.frame to be used in haplo.glm.
In the example below we prepare a genotype matrix, geno.glm, and create a data.frame object,
glm.data, for use in haplo.glm.

> # set up data for haplo.glm, include geno.glm,
> # covariates age and male, and responses resp and y.bin
> geno <- hla.demo[,c(17,18,21:24)]
> geno.glm <- setupGeno(geno, miss.val=c(0,NA), locus.label=label)
> attributes(geno.glm)

$dim
[1] 220  6

$dimnames
$dimnames[[1]]
NULL

$dimnames[[2]]
[1] "DQB.a1" "DQB.a2" "DRB.a1" "DRB.a2" "B.a1" "B.a2"
$class
[1] "model.matrix"

$unique.alleles
$unique.alleles[[1]]
[1] "21" "31" "32" "33" "42" "51" "52" "53" "61" "62" "63" "64"

$unique.alleles[[2]]
[1] "1" "2" "3" "4" "7" "8" "9" "10" "11" "13" "14"

$unique.alleles[[3]]
[1] "7" "8" "13" "14" "18" "27" "35" "37" "38" "39" "41" "42" "44" "45" "46"
[16] "47" "48" "49" "50" "51" "52" "55" "56" "57" "58" "60" "61" "62" "63" "70"

> y.bin <- 1*(resp.cat=="low")
> glm.data <- data.frame(geno.glm, age=age, male=male, y=resp, y.bin=y.bin)

6.3 Rare Haplotypes

The issue of deciding which haplotypes to use for association is critical in haplo.glm. By default it will model a rare haplotype effect so that the effects of other haplotypes are in reference to the baseline effect of the one common haplotype. The rules for choosing haplotypes to be modeled in haplo.glm are similar to the rules in haplo.score: by a minimum frequency or a minimum expected count in the sample.

Two control parameters in haplo.glm.control may be used to control this setting: haplo.freq.min may be set to a selected minimum haplotype frequency, and haplo.min.count may be set to select the cut-off for minimum expected haplotype count in the sample. The default minimum frequency cut-off in haplo.glm is set to 0.01. More discussion on rare haplotypes takes place in section 6.7.4.

6.4 Regression for a Quantitative Trait

The following illustrates how to fit a regression of a quantitative trait y on the haplotypes estimated from the geno.glm matrix, and the covariate male. For na.action, we use na.geno.keep, which keeps a subject with missing values in the genotype matrix if they are not missing all alleles, but removes subjects with missing values (NA) in either the response or covariate.

> # glm fit with haplotypes, additive gender covariate on gaussian response
> fit.gaus <- haplo.glm(y ~ male + geno.glm, family=gaussian, data=glm.data,
+     na.action="na.geno.keep", locus.label = label, x=TRUE,
+     control=haplo.glm.control(haplo.freq.min=.02))
> summary(fit.gaus)

Call:
  haplo.glm(formula = y ~ male + geno.glm, family = gaussian, data = glm.data,
             na.action = "na.geno.keep", locus.label = label, control = haplo.glm.control(haplo.freq.min=.02),
             x = TRUE)
Deviance Residuals:

<table>
<thead>
<tr>
<th>Min</th>
<th>1Q</th>
<th>Median</th>
<th>3Q</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.46945</td>
<td>-0.92052</td>
<td>-0.06533</td>
<td>0.94874</td>
<td>2.37199</td>
</tr>
</tbody>
</table>

Coefficients:

<table>
<thead>
<tr>
<th></th>
<th>coef</th>
<th>se</th>
<th>t.stat</th>
<th>pval</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>1.06436</td>
<td>0.34283</td>
<td>3.10464</td>
<td>0.002</td>
</tr>
<tr>
<td>male</td>
<td>0.09735</td>
<td>0.15521</td>
<td>0.62723</td>
<td>0.531</td>
</tr>
<tr>
<td>geno.glm.17</td>
<td>0.28022</td>
<td>0.43549</td>
<td>0.64346</td>
<td>0.521</td>
</tr>
<tr>
<td>geno.glm.34</td>
<td>-0.31713</td>
<td>0.34342</td>
<td>-0.92342</td>
<td>0.357</td>
</tr>
<tr>
<td>geno.glm.77</td>
<td>0.22167</td>
<td>0.36126</td>
<td>0.61360</td>
<td>0.540</td>
</tr>
<tr>
<td>geno.glm.78</td>
<td>1.14144</td>
<td>0.38382</td>
<td>2.97390</td>
<td>0.003</td>
</tr>
<tr>
<td>geno.glm.100</td>
<td>0.55557</td>
<td>0.36427</td>
<td>1.52517</td>
<td>0.129</td>
</tr>
<tr>
<td>geno.glm.138</td>
<td>0.98229</td>
<td>0.30329</td>
<td>3.23875</td>
<td>0.001</td>
</tr>
<tr>
<td>geno.glm.rare</td>
<td>0.39765</td>
<td>0.18191</td>
<td>2.18591</td>
<td>0.030</td>
</tr>
</tbody>
</table>

(Dispersion parameter for gaussian family taken to be 1.269581)

Null deviance: 297.01 on 219 degrees of freedom
Residual deviance: 267.88 on 211 degrees of freedom
AIC: 687.65

Number of Fisher Scoring iterations: 268

Haplotypes:

<table>
<thead>
<tr>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>hap.freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>geno.glm.17</td>
<td>21</td>
<td>7</td>
<td>44</td>
</tr>
<tr>
<td>geno.glm.34</td>
<td>31</td>
<td>4</td>
<td>44</td>
</tr>
<tr>
<td>geno.glm.77</td>
<td>32</td>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>geno.glm.78</td>
<td>32</td>
<td>4</td>
<td>62</td>
</tr>
<tr>
<td>geno.glm.100</td>
<td>51</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>geno.glm.138</td>
<td>62</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>geno.glm.rare</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>haplo.base</td>
<td>21</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Explanation of Results

The new summary function for `haplo.glm` shows much the same information as summary for `glm` objects with the extra table for the haplotype frequencies. The above table for Coefficients lists the estimated regression coefficients (coef), standard errors (se), the corresponding t-statistics (t.stat), and p-values (pval). The labels for haplotype coefficients are a concatenation of the name of the genotype matrix (`geno.glm`) and unique haplotype codes assigned within `haplo.glm`. The haplotypes corresponding to these haplotype codes are listed in the Haplotypes table, along with the estimates of the haplotype frequencies (hap.freq). The rare haplotypes, those with expected counts less than `haplo.min.count=5` (equivalent to having frequencies less than...
haplo.freq.min = 0.0113636363636364) in the above example), are pooled into a single category labeled geno.glm.rare. The haplotype chosen as the baseline category for the design matrix (most frequent haplotype is the default) is labeled as haplo.base; more information on the baseline may be found in section 6.7.2.

6.5 Fitting Haplotype x Covariate Interactions

Interactions are fit by the standard S-language model syntax, using a ‘*’ in the model formula to indicate main effects and interactions. Some other formula constructs are not supported, so use the formula parameter with caution. Below is an example of modeling the interaction of male and the haplotypes. Because more terms will be estimated in this case, we limit how many haplotypes will be included by increasing haplo.min.count to 10.

```r
> # glm fit haplotypes with covariate interaction
> fit.inter <- haplo.glm(formula = y ~ male * geno.glm,
+   family = gaussian, data=glm.data,
+   na.action="na.geno.keep",
+   locus.label = label,
+   control = haplo.glm.control(haplo.min.count = 10))
> summary(fit.inter)

Call:
  haplo.glm(formula = y ~ male * geno.glm, family = gaussian, data = glm.data,
            na.action = "na.geno.keep", locus.label = label, control = haplo.glm.control(haplo.min.count = 10))

Deviance Residuals:

          Min       1Q     Median       3Q      Max
-2.23387  -0.90661  -0.05953   0.96140  2.48859

Coefficients:

                   coef      se      t.stat  pval
(Intercept) 0.97536 0.52268  1.86607 0.063
male        0.25806 0.67351  0.38315 0.702
geno.glm.17 0.14443 0.54544  0.26479 0.791
ngen.glm.34 -0.17161 0.66773  0.25700 0.797
ngen.glm.77  0.80523 0.64951  1.23975 0.216
ngen.glm.78  0.49557 0.56574  0.87596 0.382
ngen.glm.100 0.52310 0.48067  1.08228 0.278
ngen.glm.138 1.15704 0.42325  2.73371 0.007
ngen.glm.rare 0.45547 0.28721  1.58587 0.114
male:ngen.glm.17 0.50872 0.87531  0.58119 0.562
male:ngen.glm.34 -0.28137 0.78570  0.35812 0.721
male:ngen.glm.77 -0.90084 0.79114 -1.13865 0.256
male:ngen.glm.78  1.26376 0.77131  1.63846 0.103
male:ngen.glm.100 0.05074 0.77470  0.06549 0.948
male:ngen.glm.138 -0.44587 0.61903 -0.72027 0.472
```

30
DQB DRB B hap.freq
---
| geno.glm.17 | 21  | 7  | 44  | 0.02346 |
| geno.glm.34 | 31  | 4  | 44  | 0.02845 |
| geno.glm.77 | 32  | 4  | 60  | 0.03060 |
| geno.glm.78 | 32  | 4  | 62  | 0.02413 |
| geno.glm.100| 51  | 1  | 35  | 0.03013 |
| geno.glm.138| 62  | 2  | 7   | 0.05049 |
| geno.glm.rare| *  | *  | *   | 0.70863 |
| haplo.base  | 21  | 3  | 8   | 0.10410 |

### Explanation of Results

The listed results are as explained under section 6.4. The main difference is that the interaction coefficients are labeled as a concatenation of the covariate (male in this example) and the name of the haplotype, as described above. In addition, estimates may differ because the model has changed.

### 6.6 Regression for a Binomial Trait

Next we illustrate the fitting of a binomial trait with the same genotype matrix and covariate.

```r
> # gender and haplotypes fit on binary response,
> # return model matrix
> fit.bin <- haplo.glm(y.bin ~ male + geno.glm, family = binomial,+
>                     data=glm.data, na.action = "na.geno.keep",
>                     locus.label=label,
>                     control = haplo.glm.control(haplo.min.count=10))
> summary(fit.bin)

Call:
haplo.glm(formula = y.bin ~ male + geno.glm, family = binomial,
  data = glm.data, na.action = "na.geno.keep", locus.label = label,
  control = haplo.glm.control(haplo.min.count = 10))

Deviance Residuals:
Min 1Q Median 3Q Max
-1.5559 -0.7996 -0.6473 1.0591 2.4348

Coefficients:

<table>
<thead>
<tr>
<th></th>
<th>coef</th>
<th>se</th>
<th>t.stat</th>
<th>pval</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>1.5457</td>
<td>0.6547</td>
<td>2.3610</td>
<td>0.019</td>
</tr>
<tr>
<td>male</td>
<td>-0.4802</td>
<td>0.3308</td>
<td>-1.4518</td>
<td>0.148</td>
</tr>
<tr>
<td>geno.glm.17</td>
<td>-0.7227</td>
<td>0.8011</td>
<td>-0.9022</td>
<td>0.368</td>
</tr>
<tr>
<td>geno.glm.34</td>
<td>0.3641</td>
<td>0.6798</td>
<td>0.5356</td>
<td>0.593</td>
</tr>
<tr>
<td>geno.glm.77</td>
<td>-0.9884</td>
<td>0.7328</td>
<td>-1.3489</td>
<td>0.179</td>
</tr>
<tr>
<td>geno.glm.78</td>
<td>-1.4093</td>
<td>0.8543</td>
<td>-1.6496</td>
<td>0.101</td>
</tr>
<tr>
<td>geno.glm.100</td>
<td>-2.5907</td>
<td>1.1278</td>
<td>-2.2971</td>
<td>0.023</td>
</tr>
<tr>
<td>geno.glm.138</td>
<td>-2.7156</td>
<td>0.8524</td>
<td>-3.1860</td>
<td>0.002</td>
</tr>
<tr>
<td>geno.glm.rare</td>
<td>-1.2610</td>
<td>0.3537</td>
<td>-3.5647</td>
<td>0.000</td>
</tr>
</tbody>
</table>

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 263.50 on 219 degrees of freedom
Residual deviance: 233.46 on 211 degrees of freedom
AIC: 251.11

Number of Fisher Scoring iterations: 61

Haplotypes:

<table>
<thead>
<tr>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>hap.freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>geno.glm.17</td>
<td>21</td>
<td>7</td>
<td>44</td>
</tr>
<tr>
<td>geno.glm.34</td>
<td>31</td>
<td>4</td>
<td>44</td>
</tr>
<tr>
<td>geno.glm.77</td>
<td>32</td>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>geno.glm.78</td>
<td>32</td>
<td>4</td>
<td>62</td>
</tr>
<tr>
<td>geno.glm.100</td>
<td>51</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>geno.glm.138</td>
<td>62</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>geno.glm.rare</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>haplo.base</td>
<td>21</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Explanation of Results

The underlying methods for haplo.glm are based on a prospective likelihood. Normally, this type of likelihood works well for case-control studies with standard covariates. For ambiguous haplotypes, however, one needs to be careful when interpreting the results from fitting haplo.glm to case-control data. Because cases are over-sampled, relative to the population prevalence (or incidence, for incident cases), haplotypes associated with disease will be over-represented in the case sample, and so estimates of haplotype frequencies will be biased. Positively associated haplotypes will have haplotype frequency estimates that are higher than the population haplotype frequency. To avoid this problem, one can weight each subject. The weights for the cases should be the population prevalence, and the weights for controls should be 1 (assuming the disease is rare in
the population, and controls are representative of the general population). See Stram et al.[8] for
background on using weights, and see the help file for haplo.glm for how to implement weights.

The estimated regression coefficients for case-control studies can be biased by either a large
amount of haplotype ambiguity and mis-specified weights, or by departures from Hardy-Weinberg
Equilibrium of the haplotypes in the pool of cases and controls. Generally, the bias is small, but
tends to be towards the null of no association. See Stram et al. [8] and Epstein and Satten [9] for
further details.

6.6.1 Caution on Rare Haplotypes with Binomial Response

If a rare haplotype occurs only in cases or only in controls, the fitted values would go to 0 or
1, where R would issue a warning. Also, the coefficient estimate for that haplotype would go to
positive or negative infinity. If the default haplo.min.count=5 were used above, this warning would
appear. To keep this from occurring in other model fits, increase the minimum count or minimum
frequency.

6.7 Control Parameters

Additional parameters are handled using control, which is a list of parameters providing additional
functionality in haplo.glm. This list is set up by the function haplo.glm.control. See the help file
(help(haplo.glm.control)) for a full list of control parameters, with details of their usage. Some of
the options are described here.

6.7.1 Controlling Genetic Models: haplo.effect

The haplo.effect control parameter for haplo.glm instructs whether the haplotype effects are fit as
additive, dominant, or recessive. That is, haplo.effect determines whether the covariate (x)
coding of haplotypes follows the values in Table 1 for each effect type. Heterozygous means a subject has
one copy of a particular haplotype, and homozygous means a subject has two copies of a particular
haplotype.

<table>
<thead>
<tr>
<th>Hap - Pair</th>
<th>additive</th>
<th>dominant</th>
<th>recessive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heterozygous</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Homozygous</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Note that in a recessive model, the haplotype effects are estimated only from subjects who are ho-
mozygous for a haplotype. Some of the haplotypes which meet the haplo.freq.min and haplo.count.min
cut-offs may occur as homozygous in only a few of the subjects. As stated in 5.8, recessive models
should be used when the region has multiple common haplotypes.

The default haplo.effect is additive, whereas the example below illustrates the fit of a dominant
effect of haplotypes for the gaussian trait with the gender covariate.
> # control dominant effect of haplotypes (haplo.effect)
> # by using haplo.glm.control
> fit.dom <- haplo.glm(y ~ male + geno.glm, family = gaussian,
+     data = glm.data, na.action = "na.geno.keep",
+     locus.label = label,
+     control = haplo.glm.control(haplo.effect='dominant',
+     haplo.min.count=8))
> summary(fit.dom)

Call:
  haplo.glm(formula = y ~ male + geno.glm, family = gaussian, data = glm.data,
            na.action = "na.geno.keep", locus.label = label, control = haplo.glm.control(haplo.effect = "dominant",
            haplo.min.count = 8))

Deviance Residuals:
     Min      1Q  Median      3Q     Max
-2.48099 -1.01196  0.01035  1.00557  2.48801

Coefficients:
               coef     se  t.stat     pval
(Intercept) 1.64935 0.37350  4.41593 0.000
male         0.07969 0.15726  0.50673 0.613
geno.glm.17 -0.06035 0.42317 -0.14262 0.887
geno.glm.34 -0.66499 0.36392 -1.82731 0.069
geno.glm.77 -0.07339 0.34665 -0.21171 0.833
geno.glm.78  0.85369 0.36421  2.34394 0.020
geno.glm.100 0.24697 0.34561  0.71458 0.476
geno.glm.138 0.67295 0.28163  2.38944 0.018
geno.glm.rare 0.11195 0.34006  0.32922 0.742

(Dispersion parameter for gaussian family taken to be 1.300586)

   Null deviance: 297.01  on 219  degrees of freedom
Residual deviance: 274.42  on 211  degrees of freedom
AIC: 692.96

Number of Fisher Scoring iterations: 91

Haplotypes:
  DQB DRB  B  hap.freq
geno.glm.17 21  7 44 0.02297
geno.glm.34 31  4 44 0.02855
geno.glm.77 32  4 60 0.03019
geno.glm.78 32  4 62 0.02391
geno.glm.100 51  1 35 0.03003
6.7.2 Selecting the Baseline Haplotype

The haplotype chosen for the baseline in the model is the one with the highest frequency. Sometimes the most frequent haplotype may be an at-risk haplotype, and so the measure of its effect is desired. To specify a more appropriate haplotype as the baseline in the binomial example, choose from the list of other common haplotypes, fit.bin$haplo.common. To specify an alternative baseline, such as haplotype 77, use the control parameter haplo.base and haplotype code, as in the example below.

```r
> # control baseline selection, perform the same exact run as fit.bin,
> # but different baseline by using haplo.base chosen from haplo.common
> fit.bin$haplo.common
[1] 17 34 77 78 100 138

> fit.bin$haplo.freq.init[fit.bin$haplo.common]
[1] 0.02332031 0.02848720 0.03060053 0.02349463 0.03018431 0.05097906

> fit.bin.base77 <- haplo.glm(y.bin ~ male + geno.glm, family = binomial,
+   data = glm.data, na.action = "na.geno.keep",
+   locus.label = label,
+   control = haplo.glm.control(haplo.base=77,
+                             haplo.min.count=8))
> summary(fit.bin.base77)
```

```
Call:
haplo.glm(formula = y.bin ~ male + geno.glm, family = binomial,
           data = glm.data, na.action = "na.geno.keep", locus.label = label,
           control = haplo.glm.control(haplo.base = 77, haplo.min.count = 8))

Deviance Residuals:
Min 1Q Median 3Q Max
-1.5559 -0.7996 -0.6473 1.0591 2.4348

Coefficients:
               coef   se  t.stat   pval
(Intercept) -0.4311 1.3586 -0.3173 0.751
male -0.4802 0.3308 -1.4518 0.148
geno.glm.4  0.9884 0.7328  1.3489 0.179
geno.glm.17 0.2657 1.0254  0.2591 0.796
geno.glm.34 1.3525 0.9223  1.4665 0.144
geno.glm.78 -0.4209 1.0430 -0.4035 0.687
geno.glm.100-1.6023 1.3007 -1.2319 0.219
```
Haplotypes:

<table>
<thead>
<tr>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>hap.freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>geno.glm.4</td>
<td>21</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>geno.glm.17</td>
<td>21</td>
<td>7</td>
<td>44</td>
</tr>
<tr>
<td>geno.glm.34</td>
<td>31</td>
<td>4</td>
<td>44</td>
</tr>
<tr>
<td>geno.glm.78</td>
<td>32</td>
<td>4</td>
<td>62</td>
</tr>
<tr>
<td>geno.glm.100</td>
<td>51</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>geno.glm.138</td>
<td>62</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>geno.glm.rare</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>haplo.base</td>
<td>32</td>
<td>4</td>
<td>60</td>
</tr>
</tbody>
</table>

Explanation of Results

The above model has the same haplotypes as fit.bin, except haplotype 4, the old baseline, now has an effect estimate while haplotype 77 is the new baseline. Due to randomness in the starting values of the haplotype frequency estimation, different runs of haplo.glm may result in a different set of haplotypes meeting the minimum counts requirement for being modeled. Therefore, once you have arrived at a suitable model, and you wish to modify it by changing baseline and/or effects, you can make results consistent by controlling the randomness using set.seed, as described in section 2.4. In this document, we use the same seed before making fit.bin and fit.bin.base77.

6.7.3 Rank of Information Matrix and eps.svd (NEW)

Similar to recent additions to haplo.score in section 5.7, we give the user control over the epsilon parameter determining the number of singular values when determining the rank of the information matrix in haplo.glm. Finding the generalized inverse of this matrix can be problematic when either the response variable or a covariate has a large variance and is not scaled before passed to haplo.glm. The rank of the information matrix is determined by the number of non-zero singular values a small cutoff, epsilon. When the singular values for the coefficients are on a larger numeric scale than those for the haplotype frequencies, the generalized inverse may incorrectly determine the information matrix is not of full rank. Therefore, we allow the user to specify the epsilon as eps.svd in the control parameters for haplo.glm. A simpler fix, which we strongly suggest, is for the user to pre-scale any continuous responses or covariates with a large variance.

Here we demonstrate what happens when we increase the variance of a gaussian response by
2500. We see that the coefficients are all highly significant and the rank of the information matrix is much smaller than the scaled gaussian fit.

```r
> glm.data$ybig <- glm.data$y*50
> fit.gausbig <- haplo.glm(formula = ybig ~ male + geno.glm, family = gaussian,
+   data = glm.data, na.action = "na.geno.keep", locus.label = label,
+   control = haplo.glm.control(haplo.freq.min = 0.02), x = TRUE)
> summary(fit.gausbig)
```

Call:
```
 haplo.glm(formula = ybig ~ male + geno.glm, family = gaussian,
   data = glm.data, na.action = "na.geno.keep", locus.label = label,
   control = haplo.glm.control(haplo.freq.min = 0.02), x = TRUE)
```

Deviance Residuals:

```
                       Min      1Q    Median      3Q     Max
-123.472   -46.026     -3.267   47.450  118.550
```

Coefficients:

```
                coef     se     t.stat   pval
(Intercept)  53.2180 1.7343 30.6849   0.000
  male        4.8675 6.0042  0.8107   0.418
 geno.glm.17 14.0111 0.2579 54.3195   0.000
 geno.glm.34 -15.8563 1.0033 -15.8044   0.000
 geno.glm.77  11.0835 0.9978 11.1078   0.000
 geno.glm.78  57.0720 0.3855 148.0436   0.000
 geno.glm.100 27.7784 0.3228  86.0564   0.000
 geno.glm.138 49.1143 1.0334  47.5256   0.000
geno.glm.rare 19.8824 3.4583   5.7493   0.000
```

(Dispersion parameter for gaussian family taken to be 3173.952)

Null deviance: 742530 on 219 degrees of freedom
Residual deviance: 669704 on 211 degrees of freedom
AIC: 2408.9

Number of Fisher Scoring iterations: 268

Haplotypes:

```
   DQB DRB B hap.freq
geno.glm.17  21  7 44  0.02291
geno.glm.34  31  4 44  0.02858
geno.glm.77  32  4 60  0.03022
geno.glm.78  32  4 62  0.02390
geno.glm.100  51  1 35  0.03008
```
> fit.gausbig$rank.info

[1] 175

> fit.gaus$rank.info

[1] 182

Now we set a smaller value for the eps.svd control parameter and find the fit matches the original Gaussian fit.

> fit.gausbig.eps <- haplo.glm(formula = ybig ~ male + geno.glm, family = gaussian, 
  + data = glm.data, na.action = "na.geno.keep", locus.label = label, 
  + control = haplo.glm.control(eps.svd=1e-10, haplo.freq.min = 0.02), x = TRUE)
> summary(fit.gausbig.eps)

Call:
  haplo.glm(formula = ybig ~ male + geno.glm, family = gaussian, 
  data = glm.data, na.action = "na.geno.keep", locus.label = label, 
  control = haplo.glm.control(eps.svd = 1e-10, haplo.freq.min = 0.02),
  x = TRUE)

Deviance Residuals:

  Min    1Q  Median    3Q   Max
-123.472  -46.026   -3.267   47.450  118.550

(Dispersion parameter for gaussian family taken to be 3173.952)

Null deviance: 742530 on 219 degrees of freedom
Residual deviance: 669704 on 211 degrees of freedom
AIC: 2408.9
Number of Fisher Scoring iterations: 268

Haplotypes:

<table>
<thead>
<tr>
<th>haplo.base</th>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>hap.freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>geno.glm.17</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>0.02291</td>
</tr>
<tr>
<td>geno.glm.34</td>
<td>31</td>
<td>4</td>
<td>44</td>
<td>0.02858</td>
</tr>
<tr>
<td>geno.glm.77</td>
<td>32</td>
<td>4</td>
<td>60</td>
<td>0.03022</td>
</tr>
<tr>
<td>geno.glm.78</td>
<td>32</td>
<td>4</td>
<td>62</td>
<td>0.02390</td>
</tr>
<tr>
<td>geno.glm.100</td>
<td>51</td>
<td>1</td>
<td>35</td>
<td>0.03008</td>
</tr>
<tr>
<td>geno.glm.138</td>
<td>62</td>
<td>2</td>
<td>7</td>
<td>0.05023</td>
</tr>
<tr>
<td>geno.glm.rare</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>0.71000</td>
</tr>
</tbody>
</table>

Another notable control parameter is the minimum frequency for a rare haplotype to be included in the calculations for standard error (se) of the coefficients, or `haplo.min.info`. The default value is 0.001, which means that haplotypes with frequency less than that will be part of the rare haplotype coefficient estimate, but it will not be used in the standard error calculation.

The following example demonstrates a possible result when dealing with the rare haplotype effect. We show with the hla genotype data one consequence for when this occurs. However, we make it happen by setting `haplo.freq.min` equal to `haplo.min.info`, which we advise strongly against in your analyses.

```r
> ## set haplo.freq.min and haplo.min.info to same value to show how the
> ## rare coefficient may be modeled but standard error estimate is not
> ## calculated because all haps are below haplo.min.info
> > fit.bin.rare02 <- haplo.glm(y.bin ~ geno.glm, family = binomial,
+ data = glm.data, na.action = "na.geno.keep", locus.label = label,
+ control = haplo.glm.control(haplo.freq.min=.02, haplo.min.info=.02))
> > summary(fit.bin.rare02)

Call:
  haplo.glm(formula = y.bin ~ geno.glm, family = binomial, data = glm.data,
            na.action = "na.geno.keep", locus.label = label, control = haplo.glm.control(haplo.freq.min=
            haplo.min.info = 0.02))

Deviance Residuals:
    Min      1Q  Median      3Q     Max
-1.4558  -0.7238  -0.7238  1.0382  2.3083
```
Coefficients:

<table>
<thead>
<tr>
<th></th>
<th>coef</th>
<th>se</th>
<th>t.stat</th>
<th>pval</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>1.2409</td>
<td>1.3238</td>
<td>0.9374</td>
<td>0.350</td>
</tr>
<tr>
<td>geno.glm.17</td>
<td>-0.6068</td>
<td>1.5630</td>
<td>-0.3882</td>
<td>0.698</td>
</tr>
<tr>
<td>geno.glm.34</td>
<td>0.3189</td>
<td>1.2678</td>
<td>0.2516</td>
<td>0.802</td>
</tr>
<tr>
<td>geno.glm.77</td>
<td>-1.0719</td>
<td>1.3666</td>
<td>-0.7844</td>
<td>0.434</td>
</tr>
<tr>
<td>geno.glm.78</td>
<td>-1.3593</td>
<td>4.3659</td>
<td>-0.3113</td>
<td>0.756</td>
</tr>
<tr>
<td>geno.glm.100</td>
<td>-2.3984</td>
<td>2.0878</td>
<td>-1.1487</td>
<td>0.252</td>
</tr>
<tr>
<td>geno.glm.138</td>
<td>-2.6096</td>
<td>1.5043</td>
<td>-1.7347</td>
<td>0.084</td>
</tr>
<tr>
<td>geno.glm.rare</td>
<td>-1.2233</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 263.50 on 219 degrees of freedom
Residual deviance: 235.59 on 212 degrees of freedom
AIC: 251.24

Number of Fisher Scoring iterations: 71

Haplotypes:

<table>
<thead>
<tr>
<th></th>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>hap.freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>geno.glm.17</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>0.02302</td>
</tr>
<tr>
<td>geno.glm.34</td>
<td>31</td>
<td>4</td>
<td>44</td>
<td>0.02841</td>
</tr>
<tr>
<td>geno.glm.77</td>
<td>32</td>
<td>4</td>
<td>60</td>
<td>0.03058</td>
</tr>
<tr>
<td>geno.glm.78</td>
<td>32</td>
<td>4</td>
<td>62</td>
<td>0.02353</td>
</tr>
<tr>
<td>geno.glm.100</td>
<td>51</td>
<td>1</td>
<td>35</td>
<td>0.02980</td>
</tr>
<tr>
<td>geno.glm.138</td>
<td>62</td>
<td>2</td>
<td>7</td>
<td>0.05190</td>
</tr>
<tr>
<td>geno.glm.rare</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>0.70872</td>
</tr>
<tr>
<td>haplo.base</td>
<td>21</td>
<td>3</td>
<td>8</td>
<td>0.10403</td>
</tr>
</tbody>
</table>

Explanation of Results

The above results show the standard error for the rare haplotype coefficient is “NaN”, or “Not a Number” in R, which is a consequence of having most, or all, of the rare haplotypes discarded for the standard error estimate. In other datasets there may be only a few haplotypes between haplo.min.info and haplo.freq.min, and may yield misleading results for the rare haplotype coefficient. For this reason, we recommend that any inference made on the rare haplotypes be made with caution, if at all.

7 Methods for haplo.glm (NEW)

The latest updates to haplo.stats is our work to make haplo.glm to act similar to a glm object with methods to compare and assess model fits. In this section we describe the challenges and caveats
of defining these methods for a haplo.glm object and show how to use them.

7.1 fitted.values

A challenge when defining methods for haplo.glm is that we account for the ambiguity in a person’s haplotype pair. To handle this in the glm framework, the response and non-haplotype covariates are expanded for each person with a posterior probability of the haplotype given their genotype as a weight. The returned object from haplo.glm looks somewhat like a regular glm, but the model matrix, response, and thus the fitted values, are all expanded. Users who want to work with the expanded versions of those items are welcome to access them from the returned object.

We now provide a method to get the fitted values for each person, fitted.haplo.glm. These collapsed fitted values are calculated by a weighted sum of the expanded fitted values for each person where the weights are the posterior probabilities of the person’s expanded haplotype pairs.

7.2 residuals

The residuals within the haplo.glm object are also expanded for the haplotype pairs for subjects. We provide residuals.haplo.glm to get the collapsed deviance, pearson, working, and response residuals for each person. Because we have not implemented a predict method for haplo.glm, the method does not calculate partial residuals.

7.3 vcov

We provide vcov.haplo.glm as a method to get the variance-covariance matrix of model parameters in the haplo.glm object. Unlike the standard glm object, this matrix is computed and retained in the returned object. We do this because the model parameters are the model coefficients and the haplotype frequencies, and it is computationally-intensive to compute.

We show how to get the variance matrix for all the parameters and for only the model coefficients.

```r
> varmat <- vcov(fit.gaus)
> dim(varmat)
[1] 182 182

> varmat <- vcov(fit.gaus, freq=FALSE)
> dim(varmat)
[1] 9 9

> print(varmat, digits=2)

             (Intercept)     male geno.glm.17 geno.glm.34 geno.glm.77
(Intercept)  0.118  -0.01513   -0.0674   -0.0544   -0.0526
male        -0.015   0.02409    0.0065   -0.0022   -0.0038
geno.glm.17 -0.067   0.00646    0.1897    0.0335    0.0206
geno.glm.34 -0.054  -0.00218    0.0335    0.1179    0.0226
geno.glm.77 -0.053  -0.00375    0.0206    0.0226    0.1305
```
7.4 anova and Model Comparison

We use the `anova.glm` method as a framework for `anova.haplo.glm` to allow comparisons of model fits. We limit the model comparisons to multiple nested model fits, which requires that each model to be compared is either a `haplo.glm` or `glm` fitted object. We eliminate the functionality of testing sub-models of a single fit because removal of a single covariate would require re-fitting of the reduced model to get updated coefficient and haplotype frequency estimates with a maximized log-likelihood. We decided to simplify the usage and require that all models to be compared are fully fitted.

As with the `anova.glm` method, it is difficult to check for truly nested models, so we pass the responsibility on to the user. We discuss some of the requirements.

One type of two-model comparison is between models with haplotypes (expanded subjects) and a reduced model without haplotypes. We check for the same sample size in these models by comparing the collapsed sample size from a `haplo.glm` fit to the sample size from the `glm` fit, which we remind users is only a loose check of model comparability.

The other comparison of two models in `anova.haplo.glm` is to compare two models that contain the same genotypes, and inherently the same haplotypes. This is more tricky because a subject may not have the same expanded set of possible haplotype pairs across two fits of `haplo.glm` unless the same seed is set before both fits. Even if a seed is the same, the other effects that are different between the two models will affect the haplotype frequency estimates, and may still result in a different expansion of haplotype pairs per subject. Our check of the collapsed sample size for the two models still applies with the same pitfalls, but a better assurance of model comparability is to use the same seed.

In the `haplo.glm` fit we provide the likelihood ratio test of the null model against the full model, which is the most appropriate test available for `haplo.glm` objects, but it is difficult to compare the log-likelihood across two `haplo.glm` fits. Therefore, we remain consistent with `glm` model comparison [10], and use the difference in deviance to compare models. Furthermore, we restrict the asymptotic test for model comparison to be the $\chi^2$ test for goodness of fit.

Below we show how to get the LRT from the `fit.gaus` result, then show how to compare some of the nested models fit above, including a regular `glm` fit of $y \sim male$. The anova method requires the nested model to be given first, and any anova with a `haplo.glm` object should explicitly call
anova.haplo.glm.

> fit.gaus$lrt

$lrt
[1] 22.34062

$df
[1] 8

> glmfit.gaus <- glm(y~male, family="gaussian", data=glm.data)
> anova.haplo.glm(glmfit.gaus, fit.gaus)

Analysis of Deviance Table

Model 1: y ~ male
Model 2: y ~ male + geno.glm
   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
 1    218     297.00
 2    211    267.88  7   29.114  0.001752 **
---
Signif. codes: 0 ** a ˘A¨Y***^ a˘A´Z 0.001 ** a˘A¨Y**^ a˘A´Z 0.01 ** a˘A¨Y*^ a˘A´Z 0.05 ^ a˘A¨Y.^ a˘A´Z 0.1 ^ a˘A¨Y ^ a˘A´Z 1

> anova.haplo.glm(fit.gaus, fit.inter)

Analysis of Deviance Table

Model 1: y ~ male + geno.glm
Model 2: y ~ male * geno.glm
   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
 1    211     267.88
 2    204     259.82  7    8.063  0.5017

> anova.haplo.glm(glmfit.gaus, fit.gaus, fit.inter)

Analysis of Deviance Table

Model 1: y ~ male
Model 2: y ~ male + geno.glm
Model 3: y ~ male * geno.glm
   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
 1    218     297.00
 2    211    267.88  7   29.1137  0.001804 **
 3    204     259.82  7    8.0631  0.501696
---
Signif. codes: 0 ** a ˘A¨Y***^ a˘A´Z 0.001 ** a˘A¨Y**^ a˘A´Z 0.01 ** a˘A¨Y*^ a˘A´Z 0.05 ^ a˘A¨Y.^ a˘A´Z 0.1 ^ a˘A¨Y ^ a˘A´Z 1

43
8 Extended Applications

The following functions are designed to wrap the functionality of the major functions in Haplo Stats into other useful applications.

8.1 Combine Score and Group Results: \texttt{haplo.score.merge}

When analyzing a qualitative trait, such as binary, it can be helpful to align the results from \texttt{haplo.score} with \texttt{haplo.group}. To do so, use the function \texttt{haplo.score.merge}, as illustrated in the following example:

```r
> # merge haplo.score and haplo.group results
> merge.bin <- haplo.score.merge(score.bin, group.bin)
> print(merge.bin, nlines=10)
```

<table>
<thead>
<tr>
<th>Haplotype Scores, p-values, and Frequencies By Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQB</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Explanation of Results

The first column is a row index, the next columns (3 in this example) illustrate the haplotype, the \texttt{Hap.Score} column is the score statistic and \texttt{p.val} the corresponding $\chi^2$ p-value. \texttt{Hap.Freq} is the haplotype frequency for the total sample, and the remaining columns are the estimated haplotype frequencies for each of the group levels (\texttt{y.bin} in this example). The default print method only prints results for haplotypes appearing in the \texttt{haplo.score} output. To view all haplotypes, use the print option \texttt{all.haps=TRUE}, which prints all haplotypes from the \texttt{haplo.group} output. The output is ordered by the score statistic, but the \texttt{order.by} parameter can specify ordering by haplotypes or by haplotype frequencies.

8.2 Case-Control Haplotype Analysis: \texttt{haplo.cc}

It is possible to combine the results of \texttt{haplo.score}, \texttt{haplo.group}, and \texttt{haplo.glm} for case-control data, all performed within \texttt{haplo.cc}. The function performs a score test and a glm on the same haplotypes. The parameters that determine which haplotypes are used are \texttt{haplo.min.count} and
haplo.freq.min, which are set in the control parameter, as done for haplo.glm. This is a change from previous versions, where haplo.min.count was in the parameter list for haplo.cc.

Below we run haplo.cc setting the minimum haplotype frequency at 0.02. The print results are shown, in addition to the names of the objects stored in the cc.hla result.

> # demo haplo.cc where haplo.min.count is specified
> # use geno, and this function prepares it for haplo.glm
> y.bin <- 1*(hla.demo$resp.cat=="low")
> cc.hla <- haplo.cc(y=y.bin, geno=geno, locus.label = label,
+                   control=haplo.glm.control(haplo.freq.min=.02))
> ###, em.c=haplo.em.control(iseed=10))) perhaps not needed??
> print(cc.hla, nlines=25, digits=2)

---

Global Score Statistics

---

Global-stat = 29, df = 8, p-val = 0.00029

---

Counts for Cases and Controls

---

control case
157 63

---

Haplotype Scores, p-values, Hap-Frequencies (hf), and Odds Ratios (95% CI)

---

<table>
<thead>
<tr>
<th>DQB</th>
<th>DRB</th>
<th>B</th>
<th>Hap-Score</th>
<th>p-val</th>
<th>pool.hf</th>
<th>control.hf</th>
<th>case.hf</th>
<th>glm.eff</th>
<th>OR.lower</th>
</tr>
</thead>
<tbody>
<tr>
<td>147</td>
<td>56</td>
<td>2</td>
<td>7</td>
<td>-2.103</td>
<td>0.03546</td>
<td>0.0490</td>
<td>0.0679</td>
<td>0.0159</td>
<td>0.0138</td>
</tr>
<tr>
<td>98</td>
<td>32</td>
<td>4</td>
<td>7</td>
<td>-1.393</td>
<td>0.16349</td>
<td>0.0302</td>
<td>0.0376</td>
<td>0.0089</td>
<td>0.0097</td>
</tr>
<tr>
<td>77</td>
<td>32</td>
<td>4</td>
<td>62</td>
<td>-0.496</td>
<td>0.62001</td>
<td>0.0227</td>
<td>0.0263</td>
<td>0.0079</td>
<td>0.0047</td>
</tr>
<tr>
<td>76</td>
<td>32</td>
<td>4</td>
<td>60</td>
<td>0.028</td>
<td>0.97762</td>
<td>0.0307</td>
<td>0.0315</td>
<td>0.0238</td>
<td>0.0763</td>
</tr>
<tr>
<td>16</td>
<td>21</td>
<td>7</td>
<td>44</td>
<td>1.069</td>
<td>0.28516</td>
<td>0.0212</td>
<td>0.0191</td>
<td>NA</td>
<td>Eff 0.0517</td>
</tr>
<tr>
<td>52</td>
<td>31</td>
<td>4</td>
<td>44</td>
<td>2.516</td>
<td>0.01186</td>
<td>0.0285</td>
<td>0.0150</td>
<td>0.0635</td>
<td>0.3425</td>
</tr>
<tr>
<td>11</td>
<td>21</td>
<td>3</td>
<td>8</td>
<td>3.776</td>
<td>0.00016</td>
<td>0.1042</td>
<td>0.0693</td>
<td>0.1905</td>
<td>Base NA</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>1</td>
<td>8</td>
<td>NA</td>
<td>0.0023</td>
<td>0.0033</td>
<td>NA</td>
<td>R 0.1443</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>10</td>
<td>8</td>
<td>NA</td>
<td>0.0023</td>
<td>0.0032</td>
<td>NA</td>
<td>R 0.1443</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>2</td>
<td>18</td>
<td>NA</td>
<td>0.0023</td>
<td>0.0032</td>
<td>NA</td>
<td>R 0.1443</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>2</td>
<td>7</td>
<td>NA</td>
<td>0.0023</td>
<td>0.0032</td>
<td>NA</td>
<td>R 0.1443</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>3</td>
<td>18</td>
<td>NA</td>
<td>0.0046</td>
<td>0.0067</td>
<td>NA</td>
<td>R 0.1443</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>3</td>
<td>35</td>
<td>NA</td>
<td>0.0057</td>
<td>0.0065</td>
<td>NA</td>
<td>R 0.1443</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>3</td>
<td>44</td>
<td>NA</td>
<td>0.0036</td>
<td>0.0033</td>
<td>0.0159</td>
<td>R 0.1443</td>
<td></td>
</tr>
</tbody>
</table>

45
8  21  3  49  NA  NA  0.0023  NA  NA  R  0.1443
9  21  3  57  NA  NA  0.0024  NA  NA  R  0.1443
10 21  3  70  NA  NA  0.0023  NA  NA  R  0.1443
12 21  4  62  NA  NA  0.0045  0.0064  NA  R  0.1443
13 21  7  13  NA  NA  0.0024  NA  0.0238  R  0.1443
14 21  7  18  NA  NA  0.0025  NA  NA  R  0.1443
15 21  7  35  NA  NA  0.0024  NA  0.0079  R  0.1443
17 21  7  45  NA  NA  0.0023  0.0032  NA  R  0.1443
18 21  7  50  NA  NA  0.0045  0.0032  0.0079  R  0.1443
19 21  7  57  NA  NA  0.0023  0.0064  NA  R  0.1443

OR OR.upper
147 0.072  0.38
98 0.086  0.76
78 0.058  0.72
77 0.281  1.53
76 0.318  1.32
16 0.661  3.48
52 1.318  5.07
11 1.000  NA
1  0.290  0.58
2  0.290  0.58
3  0.290  0.58
4  0.290  0.58
5  0.290  0.58
6  0.290  0.58
7  0.290  0.58
8  0.290  0.58
9  0.290  0.58
10 0.290  0.58
12 0.290  0.58
13 0.290  0.58
14 0.290  0.58
15 0.290  0.58
17 0.290  0.58
18 0.290  0.58
19 0.290  0.58

> names(cc.hla)

[1] "cc.df"     "group.count" "score.lst"  "fit.lst"      "ci.prob"
[6] "exclude.subj"

Explanation of Results

First, from the names function we see that cc.hla also contains score.lst and fit.lst, which are the haplo.score and haplo.glm objects, respectively. For the printed results of haplo.cc, first are the
global statistics from haplo.score, followed by cell counts for cases and controls. The last portion of the output is a data frame containing combined results for individual haplotypes:

- **Hap-Score**: haplotype score statistic
- **p-val**: haplotype score statistic p-value
- **sim p-val**: (if simulations performed) simulated p-value for the haplotype score statistic
- **pool.hf**: haplotype frequency for the pooled sample
- **control.hf**: haplotype frequencies for the control sample only
- **case.hf**: haplotype frequencies for the case sample only
- **glm.eff**: one of three ways the haplotype appeared in the glm model: *Eff*: modeled as an effect; *Base*: part of the baseline; and *R*: a rare haplotype, included in the effect of pooled rare haplotypes
- **OR.lower**: Odds Ratio confidence interval lower limit
- **OR**: Odds Ratio for each effect in the model
- **OR.upper**: Odds Ratio confidence interval upper limit

Significance levels are indicated by the p-values for the score statistics, and the odds ratio (OR) confidence intervals for the haplotype effects. Note that the Odds Ratios are effect sizes of haplotypes, assuming haplotype effects are multiplicative. Since this last table has many columns, lines are wrapped in the output in this manual. You can align wrapped lines by the haplotype code which appears on the far left. Alternatively, instruct the print function to only print digits significant digits, and set the width settings for output in your session using the `options()` function.

### 8.3 Score Tests on Sub-Haplotypes: haplo.score.slide

To evaluate the association of sub-haplotypes (subsets of alleles from the full haplotype) with a trait, the user can evaluate a "window" of alleles by haplo.score, and slide this window across the entire haplotype. This procedure is implemented by the function `haplo.score.slide`. To illustrate this method, we use all 11 loci in the demo data, `hla.demo`.

First, make the geno matrix and the locus labels for the 11 loci. Then use `haplo.score.slide` for a window of 3 loci (n.slide=3), which will slide along the haplotype for all 9 contiguous subsets of size 3, using the previously defined gaussian trait `resp`.

```r
> # haplo.score on 11 loci, slide on 3 consecutive loci at a time
> geno.11 <- hla.demo[, -c(1:4)]
> label.11 <- c("DPB","DPA","DMA","DMB","TAP1","TAP2","DQB","DQA","DRB","B","A")
> score.slide.gaus <- haplo.score.slide(hla.demo$resp, geno.11, trait.type = + "gaussian", n.slide=3, min.count=5, locus.label=label.11)
> print(score.slide.gaus)
```
<table>
<thead>
<tr>
<th>start.loc</th>
<th>score.global.p</th>
<th>global.p.sim</th>
<th>max.p.sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.21550</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.09366</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.39042</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.48771</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0.13747</td>
<td>NA</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0.14925</td>
<td>NA</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0.11001</td>
<td>NA</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.00996</td>
<td>NA</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0.04255</td>
<td>NA</td>
</tr>
</tbody>
</table>

Explanation of Results

The first column is the row index of the nine calls to `haplo.score`, the second column is the number of the starting locus of the sub-haplotype, the third column is the global score statistic p-value for each call. The last two columns are the simulated p-values for the global and maximum score statistics, respectively. If you specify `simulate=TRUE` in the function call, the simulated p-values would be present.

8.3.1 Plot Results from `haplo.score.slide`

The results from `haplo.score.slide` can be easily viewed in a plot shown in Figure 2 below. The x-axis has tick marks for each locus, and the y-axis is the $-\log_{10}(pval)$. To select which p-value to plot, use the parameter `pval`, with choices "global", "global.sim", and "max.sim" corresponding to p-values described above. If the simulated p-values were not computed, the default is to plot the global p-values. For each p-value, a horizontal line is drawn at the height of $-\log_{10}(pval)$ across the loci over which it was calculated. For example, the p-value $score.global.p = 0.009963$ for loci 8-10 is plotted as a horizontal line at $y = 2.002$ spanning the 8th, 9th, and 10th x-axis tick marks.
> # plot global p-values for sub-haplotypes from haplo.score.slide
> plot.haplo.score.slide(score.slide.gaus)

Figure 2: Global p-values for sub-haplotypes; Gaussian Response
8.4 Scanning Haplotypes Within a Fixed-Width Window: haplo.scan

Another method to search for a candidate locus within a genome region is haplo.scan, an implementation of the method proposed in Cheng et al. 2005 [11]. This method searches for a region for which the haplotypes have the strongest association with a binary trait by sliding a window of fixed width over each marker locus, and then scans over all haplotype lengths within each window. This latter step, scanning over all possible haplotype lengths within a window, distinguishes haplo.scan from haplo.score.slide (which considers only the maximum haplotype length within a window). To account for unknown linkage phase, the function haplo.em is called prior to scanning, to create a list of haplotype pairs and posterior probabilities. To illustrate the scanning window, consider a 10-locus dataset. When placing a window of width 3 over locus 5, the possible haplotype lengths that contain locus 5 are three (loci 3-4-5, 4-5-6, and 5-6-7), two (loci 4-5 and 5-6) and one (locus 5). For each of these loci subsets a score statistic is computed, which is based on the difference between the mean vector of haplotype counts for cases and that for controls. The maximum of these score statistics, over all possible haplotype lengths within a window, is the locus-specific test statistic, or the locus scan statistic. The global test statistic is the maximum over all computed score statistics. To compute p-values, the case/control status is randomly permuted. Below we run haplo.scan on the 11-locus HLA dataset with a binary response and a window width of 3, but first we use the results of summaryGeno to choose subjects with less than 50,000 haplotype pairs to speed calculations with all 11 polymorphic loci with many missing alleles.

```r
> geno.11 <- hla.demo[, -c(1:4)]
> y.bin <- 1*(hla.demo$resp.cat == "low")
> hla.summary <- summaryGeno(geno.11, miss.val = c(0, NA))
> # track those subjects with too many possible haplotype pairs (> 50,000)
> many.haps <- (1:length(y.bin))[hla.summary[,4] > 50000]
> # For speed, or even just so it will finish, make y.bin and geno.scan
> # for genotypes that don't have too many ambiguous haplotypes
> geno.scan <- geno.11[-many.haps,]
> y.scan <- y.bin[-many.haps]
> # scan haplotypes for regions within width of 3 for each locus.
> # test statistic measures difference in haplotype counts in cases and controls
> # p-values are simulated for each locus and the maximum statistic,
> # we do 100 simulations here, should use default settings for analysis
> scan.hla <- haplo.scan(y.scan, geno.scan, width = 3,
  + sim.control = score.sim.control(min.sim = 100, max.sim = 100),
  + em.control = haplo.em.control())
> print(scan.hla)
```

Call:

```r
haplo.scan(y = y.scan, geno = geno.scan, 
  width = 3, em.control = haplo.em.control(),
  sim.control = score.sim.control(min.sim = 100, 
  max.sim = 100))
```
### Locus Scan-statistic Simulated P-values

<table>
<thead>
<tr>
<th>loc-1</th>
<th>loc-2</th>
<th>loc-3</th>
<th>loc-4</th>
<th>loc-5</th>
<th>loc-6</th>
<th>loc-7</th>
<th>loc-8</th>
<th>loc-9</th>
<th>loc-10</th>
<th>loc-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim.p-val</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Loci with max scan statistic: 2
Max-Stat Simulated Global p-value: 0
Number of Simulations: 100

### Explanation of Results

In the output we report the simulated p-values for each locus test statistic. Additionally, we report the loci (or locus) which provided the maximum observed test statistic, and the Max-Stat Simulated Global p-value is the simulated p-value for that maximum statistic. We print the number of simulations, because they are performed until p-value precision criteria are met, as described in section 5.9. We would typically allow simulations to run under default parameters rather than limiting to 100 by the control parameters.

#### 8.5 Sequential Haplotype Scan Methods: seqhap

Another approach for choosing loci for haplotype associations is by seqhap, as described in Yu and Schaid, 2007 [12]. The seqhap method performs three tests for association of a binary trait over a set of bi-allelic loci. When evaluating each locus, loci close to it are added in a sequential manner based on the Mantel-Haenszel test [13]. For each marker locus, three tests are provided:

- **single locus**, the traditional single-locus $\chi^2$ test of association,
- **sequential haplotype**, based on a haplotype test for sequentially chosen loci,
- **sequential sum**, based on the sum of a series of conditional $\chi^2$ statistics.

All three tests are assessed for significance with permutation p-values, in addition to the asymptotic p-value. The point-wise p-value for a statistic at a locus is the fraction of times that the statistic for the permuted data is larger than that for the observed data. The regional p-value is the chance of observing a permuted test statistic, maximized over a region, that is greater than that for the observed data.

Similar to the permutation p-values in haplo.score as described in section 5.9, permutations are performed until a precision threshold is reached for the regional p-values. A minimum and maximum number of permutations specified in the sim.control parameter list ensure a certain accuracy is met for every simulation p-value, yet having a limit to avoid infinite run-time.

Below is an example of using seqhap on data with case-control response for a chromosome region. First set up the binary response, $y$, with 0=control, 1=case, then a genotype matrix with two columns per locus, and a vector of chromosome positions. The genotype data is available in the seqhap.dat dataset while the chromosome positions are in seqhap.pos. The following example runs seqhap with default settings for permutations and threshold parameters.
> # define binary response and genotype matrix
> data(seqhap.dat)
> data(seqhap.pos)
> y <- seqhap.dat$disease
> geno <- seqhap.dat[, -1]
> # get vector with chrom position
> pos <- seqhap.pos$pos
> seqhap.out <- seqhap(y = y, geno = geno, pos = pos, miss.val = c(0, NA),
> + r2.threshold = .95, mh.threshold = 3.84)
> seqhap.out$n.sim

[1] 4973

> print(seqhap.out)

==============================================================================

Single-locus Chi-square Test

==============================================================================

Regional permuted P-value based on single-locus test is 0.13191

<table>
<thead>
<tr>
<th>loc</th>
<th>chi.stat</th>
<th>perm.point.p</th>
<th>asym.point.p</th>
</tr>
</thead>
<tbody>
<tr>
<td>loc-1</td>
<td>1.22062</td>
<td>0.27729741</td>
<td>0.26924</td>
</tr>
<tr>
<td>loc-2</td>
<td>1.35462</td>
<td>0.23245526</td>
<td>0.24447</td>
</tr>
<tr>
<td>loc-3</td>
<td>5.20288</td>
<td>0.02010859</td>
<td>0.02255</td>
</tr>
<tr>
<td>loc-4</td>
<td>3.36348</td>
<td>0.05972250</td>
<td>0.06666</td>
</tr>
<tr>
<td>loc-5</td>
<td>3.55263</td>
<td>0.06153227</td>
<td>0.05945</td>
</tr>
<tr>
<td>loc-6</td>
<td>0.39263</td>
<td>0.53026342</td>
<td>0.53092</td>
</tr>
<tr>
<td>loc-7</td>
<td>5.54913</td>
<td>0.01829881</td>
<td>0.01849</td>
</tr>
<tr>
<td>loc-8</td>
<td>3.74740</td>
<td>0.05469535</td>
<td>0.05289</td>
</tr>
<tr>
<td>loc-9</td>
<td>0.03602</td>
<td>0.85682687</td>
<td>0.84947</td>
</tr>
<tr>
<td>loc-10</td>
<td>1.99552</td>
<td>0.17313493</td>
<td>0.15777</td>
</tr>
</tbody>
</table>

==============================================================================

Sequential Scan

==============================================================================

Loci Combined in Sequential Analysis

<table>
<thead>
<tr>
<th>seq-loc</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-loc-2</td>
<td>2 3 4 5</td>
</tr>
<tr>
<td>seq-loc-3</td>
<td>3 4 5</td>
</tr>
<tr>
<td>seq-loc-4</td>
<td>4 3</td>
</tr>
<tr>
<td>seq-loc-5</td>
<td>5</td>
</tr>
<tr>
<td>seq-loc-6</td>
<td>6 7</td>
</tr>
<tr>
<td>seq-loc-7</td>
<td>7</td>
</tr>
<tr>
<td>seq-loc-8</td>
<td>8</td>
</tr>
<tr>
<td>seq-loc-9</td>
<td>9</td>
</tr>
<tr>
<td>seq-loc-10</td>
<td>10</td>
</tr>
</tbody>
</table>

52
Sequential Haplotype Test

Regional permuted P-value based on sequential haplotype test is 0.016489

<table>
<thead>
<tr>
<th>seq-loc-1</th>
<th>hap.stat</th>
<th>df</th>
<th>perm.point.p</th>
<th>asym.point.p</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.22062</td>
<td>1</td>
<td>0.310878745</td>
<td>0.26924</td>
</tr>
<tr>
<td></td>
<td>24.16488</td>
<td>12</td>
<td>0.027950935</td>
<td>0.01932</td>
</tr>
<tr>
<td></td>
<td>19.78808</td>
<td>6</td>
<td>0.005228232</td>
<td>0.00302</td>
</tr>
<tr>
<td></td>
<td>14.95765</td>
<td>3</td>
<td>0.003016288</td>
<td>0.00185</td>
</tr>
<tr>
<td></td>
<td>3.55263</td>
<td>1</td>
<td>0.096722300</td>
<td>0.05945</td>
</tr>
<tr>
<td></td>
<td>5.45723</td>
<td>2</td>
<td>0.114216771</td>
<td>0.06531</td>
</tr>
<tr>
<td></td>
<td>5.54913</td>
<td>1</td>
<td>0.038608486</td>
<td>0.01849</td>
</tr>
<tr>
<td></td>
<td>3.74740</td>
<td>1</td>
<td>0.103961392</td>
<td>0.05289</td>
</tr>
<tr>
<td></td>
<td>0.03602</td>
<td>1</td>
<td>0.867886588</td>
<td>0.84947</td>
</tr>
<tr>
<td></td>
<td>1.99552</td>
<td>1</td>
<td>0.219384677</td>
<td>0.15777</td>
</tr>
</tbody>
</table>

Sequential Sum Test

Regional permuted P-value based on sequential sum test is 0.0032174

<table>
<thead>
<tr>
<th>seq-loc-1</th>
<th>sum.stat</th>
<th>df</th>
<th>perm.point.p</th>
<th>asym.point.p</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.22062</td>
<td>1</td>
<td>0.310878745</td>
<td>0.26924</td>
</tr>
<tr>
<td></td>
<td>21.15360</td>
<td>4</td>
<td>0.0008043435</td>
<td>0.00030</td>
</tr>
<tr>
<td></td>
<td>18.65769</td>
<td>3</td>
<td>0.0008043435</td>
<td>0.00032</td>
</tr>
<tr>
<td></td>
<td>14.61897</td>
<td>2</td>
<td>0.0020108586</td>
<td>0.00067</td>
</tr>
<tr>
<td></td>
<td>3.55263</td>
<td>1</td>
<td>0.1033581339</td>
<td>0.05945</td>
</tr>
<tr>
<td></td>
<td>5.43826</td>
<td>2</td>
<td>0.1150211140</td>
<td>0.06593</td>
</tr>
<tr>
<td></td>
<td>5.54913</td>
<td>1</td>
<td>0.0386084858</td>
<td>0.01849</td>
</tr>
<tr>
<td></td>
<td>3.74740</td>
<td>1</td>
<td>0.1041624774</td>
<td>0.05289</td>
</tr>
<tr>
<td></td>
<td>0.03602</td>
<td>1</td>
<td>0.8678865876</td>
<td>0.84947</td>
</tr>
<tr>
<td></td>
<td>1.99552</td>
<td>1</td>
<td>0.2193846773</td>
<td>0.15777</td>
</tr>
</tbody>
</table>

Explanation of Results

The output from this example first shows \textit{n.sim}, the number of permutations needed for precision on the regional p-values. Next, in the printed results, the first section (\textit{Single-locus Chi-square Test}) shows a table with columns for single-locus tests. The table includes test statistics, permuted p-values, and asymptotic p-values based on a $\chi^2$ distribution. The second section (\textit{Sequential Scan}) shows which loci are combined for association. In this example, the table shows the first locus is not combined with other loci, whereas the second locus is combined with loci 3, 4, and 5. The third section (\textit{Sequential Haplotype Test}), shows the test statistics for the sequential haplotype method with degrees of freedom and permuted and asymptotic p-values. The fourth section (\textit{Sequential Sum Test}) shows similar information for the sequential sum tests.
8.5.1 Plot Results from seqhap

The results from *seqhap* can be viewed in a useful plot shown in Figure 3. The plot is similar to the plot for *haplo.score.slide* results, with the x-axis having tick marks for all loci and the y-axis is the -log10() of p-value for the tests performed. For the sequential result for each locus, a horizontal line at the height of -log10(p-value) is drawn across the loci combined. The start locus is indicated by a filled triangle and other loci combined with the start locus are indicated by an asterisk or circle. The choices for pval include "hap" (sequential haplotype asymptotic p-value), "hap.sim" (sequential haplotype simulated p-value), "sum" (sequential sum asymptotic p-value), and "sum.sim" (sequential sum simulated p-value). The other parameter option is single, indicating whether to plot a line for the single-locus tests.
Figure 3: Plot p-values for sequential haplotype scan and single-locus tests
8.6 Creating Haplotype Effect Columns: \texttt{haplo.design}

In some instances, the desired model for haplotype effects is not possible with the methods given in \texttt{haplo.glm}. Examples include modeling just one haplotype effect, or modeling an interaction of haplotypes from different chromosomes, or analyzing censored data. To circumvent these limitations, we provide a function called \texttt{haplo.design}, which will set up an expected haplotype design matrix from a \texttt{haplo.em} object, to create columns that can be used to model haplotype effects in other modeling functions.

The function \texttt{haplo.design} first creates a design matrix for all pairs of haplotypes over all subjects, and then uses the posterior probabilities to create a weighted average contribution for each subject, so that the number of rows of the final design matrix is equal to the number of subjects. This is sometimes called the expectation-substitution method, as proposed by Zaykin et al. 2002 [4], and using this haplotype design matrix in a regression model is asymptotically equivalent to the score statistics from \texttt{haplo.score} (Xie and Stram 2005 [14]). Although this provides much flexibility, by using the design matrix in any type of regression model, the estimated regression parameters can be biased toward zero (see Lin and Zeng, 2006 [15] for concerns about the expectation-substitution method).

In the first example below, using default parameters, the returned data.frame contains a column for each haplotype that meets a minimum count in the sample \texttt{min.count}. The columns are named by the code they are assigned in \texttt{haplo.em}.

```r
> # create a matrix of haplotype effect columns from haplo.em result
> hap.effect.frame <- haplo.design(save.em)
> names(hap.effect.frame)
[1] "hap.4"  "hap.13"  "hap.17"  "hap.34"  "hap.50"  "hap.55"  "hap.69"
[8] "hap.77"  "hap.78"  "hap.99"  "hap.100"  "hap.102"  "hap.138"  "hap.140"
[15] "hap.143"  "hap.155"  "hap.162"  "hap.165"
> hap.effect.frame[1:10,1:8]

<table>
<thead>
<tr>
<th></th>
<th>hap.4</th>
<th>hap.13</th>
<th>hap.17</th>
<th>hap.34</th>
<th>hap.50</th>
<th>hap.55</th>
<th>hap.69</th>
<th>hap.77</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.1253</td>
<td>0.8747</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.2862</td>
<td>0.7138</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Additionally, \texttt{haplo.design} gives the user flexibility to make a more specific design matrix with the following parameters:

- \textbf{hapcodes}: codes assigned in the \texttt{haplo.em} object, the only haplotypes to be made into effects
- **haplo.effect**: the coding of haplotypes as additive, dominant, or recessive
- **haplo.base**: code for the baseline haplotype
- **min.count**: minimum haplotype count

This second example below creates columns for specific haplotype codes that were most interesting in `score.gaus.add`, haplotypes with alleles 21-3-8 and 62-2-7, corresponding to codes 4 and 138 in `haplo.em`, respectively. Assume we want to test their individual effects when they are coded with `haplo.effect="dominant"`.

```r
> # create haplotype effect cols for haps 4 and 138
> hap4.hap138.frame <- haplo.design(save.em, hapcodes=c(4,138),
+   haplo.effect="dominant")
> hap4.hap138.frame[1:10,]
   hap.4 hap.138
1       0    0.000000
2       0    0.8746766
3       0    0.0000000
4       0    0.0000000
5       0    0.0000000
6       1    0.0000000
7       0    1.0000000
8       0    0.0000000
9       0    0.1358696
10      0    0.0000000
```

```r
data.frame(resp, male, age,
+   hap.4=hap4.hap138.frame$hap.4,
+   hap.138=hap4.hap138.frame$hap.138)
glm.hap4.hap138 <- glm(resp ~ male + age + hap.4 + hap.138,
+   family="gaussian", data=dat.glm)
> summary(glm.hap4.hap138)
```

Call:
```r
glm(formula = resp ~ male + age + hap.4 + hap.138, family = "gaussian",
   data = dat.glm)
```

Deviance Residuals:
```r
  Min 1Q Median 3Q Max
-2.32614 -1.07489 -0.06559 1.04483 2.39044
```

Coefficients:
```r
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.913834   0.229577  8.336 9.11e-15 ***
male        0.048588   0.155290   0.313    0.7547
```

57
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>-0.002651</td>
<td>0.011695</td>
<td>-0.227</td>
<td>0.8209</td>
</tr>
<tr>
<td>hap.4</td>
<td>-0.405530</td>
<td>0.195857</td>
<td>-2.071</td>
<td>0.0396 *</td>
</tr>
<tr>
<td>hap.138</td>
<td>0.584480</td>
<td>0.261763</td>
<td>2.233</td>
<td>0.0266 *</td>
</tr>
</tbody>
</table>

---

Signif. codes: 0 ^ a,Y***a,A 0.001 ^ a,Y**a,A 0.01 ^ a,Y*a,A 0.05 ^ a,Y.a,A 0.1 ^ a,Y a,A 1

(Dispersion parameter for gaussian family taken to be 1.318277)

Null deviance: 297.01 on 219 degrees of freedom
Residual deviance: 283.43 on 215 degrees of freedom
AIC: 692.07

Number of Fisher Scoring iterations: 2
9 License and Warranty

License:

Copyright 2003 Mayo Foundation for Medical Education and Research.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to
Free Software Foundation, Inc.
59 Temple Place, Suite 330
Boston, MA 02111-1307 USA

For other licensing arrangements, please contact Daniel J. Schaid.
Daniel J. Schaid, Ph.D.
Division of Biostatistics
Harwick Building - Room 775
Mayo Clinic
200 First St., SW
Rochester, MN 55905
phone: 507-284-0639
fax: 507-284-9542
email: schaid@mayo.edu

10 Acknowledgements

This research was supported by United States Public Health Services, National Institutes of Health: Contract grant numbers R01 DE13276, R01 GM 65450, N01 AI45240, and R01 2AI33144. The hla.demo data is kindly provided by Gregory A. Poland, M.D. and the Mayo Vaccine Research Group for illustration only, and may not be used for publication.
Appendix

A Counting Haplotype Pairs When Marker Phenotypes Have Missing Alleles

The following describes the process for counting the number of haplotype pairs that are consistent with a subject’s observed marker phenotypes, allowing for some loci with missing data. Note that we refer to marker phenotypes, but our algorithm is oriented towards typical markers that have a one-to-one correspondence with their genotypes. We first describe how to count when none of the loci have missing alleles, and then generalize to allow loci to have either one or two missing alleles. When there are no missing alleles, note that homozygous loci are not ambiguous with respect to the underlying haplotypes, because at these loci the underlying haplotypes will not differ if we interchange alleles between haplotypes. In contrast, heterozygous loci are ambiguous, because we do not know the haplotype origin of the distinguishable alleles (i.e., unknown linkage phase). However, if there is only one heterozygous locus, then it doesn’t matter if we interchange alleles, because the pair of haplotypes will be the same. In this situation, if parental origin of alleles were known, then interchanging alleles would switch parental origin of haplotypes, but not the composition of the haplotypes. Hence, ambiguity arises only when there are at least two heterozygous loci. For each heterozygous locus beyond the first one, the number of possible haplotypes increases by a factor of 2, because we interchange the two alleles at each heterozygous locus to create all possible pairs of haplotypes. Hence, the number of possible haplotype pairs can be expressed as $2^x$, where $x = H - 1$, if $H$ (the number of heterozygous loci) is at least 2, otherwise $x = 0$.

Now consider a locus with missing alleles. The possible alleles at a given locus are considered to be those that are actually observed in the data. Let $a_i$ denote the number of distinguishable alleles at the locus. To count the number of underlying haplotypes that are consistent with the observed and missing marker data, we need to enumerate all possible genotypes for the loci with missing data, and consider whether the imputed genotypes are homozygous or heterozygous.

To develop our method, first consider how to count the number of genotypes at a locus, say the $i^{th}$ locus, when either one or two alleles are missing. This locus could have either a homozygous or heterozygous genotype, and both possibilities must be considered for our counting method. If the locus is considered as homozygous, and there is one allele missing, then there is only one possible genotype; if there are two alleles missing, then there are $a_i$ possible genotypes. A function to perform this counting for homozygous loci is denoted $f(a_i)$. If the locus is considered as heterozygous, and there is one allele missing, then there are $a_i - 1$ possible genotypes; if there are two alleles missing, then there are $\frac{a_i(a_i - 1)}{2}$ possible genotypes. A function to perform this counting for heterozygous loci is denoted $g(a_i)$. These functions and counts are summarized in Table A.1.

Table A.1: Factors for when a locus having missing allele(s) is counted as homozygous($f()$) or heterozygous($g()$)

Now, to use these genotype counting functions to determine the number of possible haplotype pairs, first consider a simple case where only one locus, say the $i^{th}$ locus, has two missing alleles. Suppose that the phenotype has $H$ heterozygous loci ($H$ is the count of heterozygous loci among those without missing data). We consider whether the locus with missing data is either homozygous or heterozygous, to give the count of possible haplotype pairs as
where again $x = H - 1$ if $H$ is at least 2, otherwise $x = 0$. This special case can be represented by our more general genotype counting functions as

$$f(a_i) 2^x + g(a_i) 2^{x+1}$$

When multiple loci have missing data, we need to sum over all possible combinations of heterozygous and homozygous genotypes for the incomplete loci. The rows of Table A.2 below present these combinations for up to $m = 3$ loci with missing data. Note that as the number of heterozygous loci increases (across the columns of Table A.2), so too does the exponent of 2. To calculate the total number of pairs of haplotypes, given observed and possibly missing genotypes, we need to sum the terms in Table A.2 across the appropriate row. For example, with $m = 3$, there are eight terms to sum over. The general formulation for this counting method can be expressed as

$$TotalPairs = \sum_{j=0}^{m} \sum_{combo} C(combo, j)$$

where combo is a particular pattern of heterozygous and homozygous loci among the loci with missing values (e.g., for $m = 3$, one combination is the first locus heterozygous and the 2\textsuperscript{nd} and 3\textsuperscript{rd} third as homozygous), and $C(combo, j)$ is the corresponding count for this pattern when there are $i$ loci that are heterozygous (e.g., for $m = 3$ and $j = 1$, as illustrated in Table A.2).

### Table A.2: Genotype counting terms when $m$ loci have missing alleles, grouped by number of heterozygous loci (out of $m$)

<table>
<thead>
<tr>
<th>$m$</th>
<th>$j = 0 \text{of } m$</th>
<th>$j = 1 \text{of } m$</th>
<th>$j = 2 \text{of } m$</th>
<th>$j = 3 \text{of } m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$2^x$</td>
<td>$g(a_i) 2^{x+1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$f(a_1) 2^x$</td>
<td>$g(a_1)f(a_2) 2^{x+1}$</td>
<td>$g(a_1)g(a_2) 2^{x+1}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$f(a_1)f(a_2) 2^x$</td>
<td>$g(a_1)f(a_2) 2^{x+1}$</td>
<td>$g(a_1)g(a_2) 2^{x+1}$</td>
<td>$g(a_1)g(a_2)g(a_3) 2^{x+2}$</td>
</tr>
<tr>
<td>3</td>
<td>$f(a_1)f(a_2)f(a_3) 2^x$</td>
<td>$g(a_1)f(a_2)f(a_3) 2^{x+1}$</td>
<td>$g(a_1)g(a_2)f(a_3) 2^{x+2}$</td>
<td>$g(a_1)g(a_2)g(a_3) 2^{x+2}$</td>
</tr>
</tbody>
</table>
References


