Haplo Stats User Manual

Version 1.1.0

Statistical Methods for Haplotypes when

Linkage Phase is Ambiguous

Jason P. Sinnwell and Daniel J. Schaid

Mayo Clinic
Rochester, MN

Contact Information:

Send feedback and questions by email to Jason Sinnwell, sinnwell@mayo.edu.

Revision Date: Wednesday, March 10, 2004

Table of Contents

1 Brief Description

Haplo Stats is a suite of S-PLUS/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). The genetic markers are assumed to be codominant (i.e., one-to-one correspondence between their genotypes and their phenotypes), and so we refer to the measurements of genetic markers as genotypes. The main functions in Haplo Stats are:

`haplo.em`
..... for the estimation of haplotype frequencies, and posterior probabilities of haplotype pairs for a subject, conditional on the observed marker data

`haplo.glm`
..... glm regression models for the regression of a trait on haplotypes, possibly including covariates and interactions

`haplo.score`
..... score statistics to test associations between haplotypes and a wide variety of traits, including binary, ordinal, quantitative, and Poisson.

IMPORTANT NOTE: If you have used the previously distributed package for `haplo.score`, it is important to note that the function `haplo.score` has changed dramatically from the previous distribution, including the parameters passed to this function. Please follow the examples provided in this document to see how to use this function.

2 Operating System and Installation

Haplo Stats version 1.1.0 library is written for both S-PLUS version 6.0 and R version 1.7.1 on a Unix operating system. Installation procedures for S-PLUS and R systems are slightly different, and explained in the included `README.haplo.stat` file. The procedures for running analyses are the same for S-PLUS and R.

3 New Features

1. **Accounting for missing genotypes**: The first version (1.0) of `haplo.score` removed subjects who were missing any marker genotypes. The current Haplo Stats functions allow for missing marker genotypes.

2. **Improved EM algorithm for estimating haplotype frequencies** (see section 6.1).

3. **Haplotype frequencies by subsets:** Another new feature provides estimated haplotype frequencies for subsets defined by levels of a qualitative ”group” variable (see the new function `haplo.group`). This information can be combined with output from `haplo.score` by the

new function `haplo.score.merge`. These new functions are useful for case-control studies in order to align estimates of haplotype frequencies for cases and controls with the corresponding score statistics.

4. Regression models: The function `haplo.glm` is a major new addition, which provides a way to regress a trait on haplotypes, covariates, and possibly their interactions. Earlier versions of `haplo.glm` did not work in R, but this is now corrected by use of a new `setupGeno` function.

4 Getting Started

After installing the Haplo Stats package, the routines and an example data set are available by starting an S-PLUS or R session and attaching the appropriate directory. The easiest way to get started is by following an example. An experienced user may want to skip the example and simply view the details in the help files. As illustrated in the following example session, a user enters the indented text following the prompt ">", and the output results follow.

Example Data

First attach the directory with the Haplo Stats routines and example data set (`hla.demo`). To do this, use the local library concept, and type

```
> library(haplo.stats, lib.loc="/Installation Directory Path Here/")
> setupData(hla.demo)
```

where "/`Installation Directory Path Here`/" is the directory path that leads to the directory `haplo.stats`. For R users, the following examples are fully automated for R by typing

```
> demo(UserManTest)
```

Note it will take a long time to run through all the examples automatically, so you may prefer to step through each one as illustrated below.

Look at the names of variables in `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"
```

The variables 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`	sex code with 1="male" , 0="female"
`age`	age (months) at immunization

The remaining variables 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 to attach hla.demo, so the variables can be accessed by simply typing their names.

```
> attach(hla.demo)
```

Creating a geno matrix

Many of the functions require a matrix of genotypes, denoted here 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,a,B,b,C,c). 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.

```
> geno <- hla.demo[,c(17,18,21:24)]
```

Create some labels for the loci.

```
> label <-c("DQB","DRB","B")
```

Random Numbers and Setting Seed

Simulations are used in several of the functions (e.g., to determine random starting values for haplo.em, and to compute permutation p-values in haplo.score). In order to reproduce results in this user guide, you must set the .Random.seed before any function which uses random numbers. We illustrate this below, and it appears throughout the script UserManTest.q, but we do not illustrate it throughout this user guide, to avoid clutter. In practice, the user would not ordinarily reset the seed.

```
seed <- c(17, 53, 1, 40, 37, 0, 62, 56, 5, 52, 12, 1)
set.seed(seed)
```

Different Random Numbers for S-PLUS and R

The above mechanism for controlling .Random.seed makes results reproducible in the respective S-PLUS and R platforms. However, the random number generators for S-PLUS and R use the seeds differently, so results will not completely agree across platforms. Because the results in this document were generated by S-PLUS on a Unix platform, results from R that depend on random numbers will not exactly match the results in this document. None the less, results can be forced to agree across platforms by omitting the randomness within haplo.em by setting the control parameter n.try=1 within haplo.em.control (see section 6.3).

5 Preview for Missing Data by "summaryGeno"

Before running the haplotype statistics, the user may want to look for 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, and the user may want to remove some of the subjects with a lot of missing data. This can be accomplished with the function summaryGeno. This function checks for missing allele information and counts the number of potential haplotype pairs that are consistent with the observed data (see Appendix for a description of this counting algorithm).

IMPORTANT: Coding Missing Alleles

The codes for missing values of alleles are defined by the parameter miss.val, which may be a vector to define multiple missing value codes. Because it has been common practice to use a zero "0" to code for missing alleles, the default values for miss.val are 0 and NA.

```
> geno.desc <- summaryGeno(geno, miss.val=c(0,NA))

> print(geno.desc)

     loc miss-0 loc miss-1 loc miss-2 num_enum_rows
1           3           0           0             4
2           3           0           0             4
3           3           0           0             4
.
.
.
80          3           0           0             4
81          2           0           1          1800
82          3           0           0             2
83          3           0           0             1
.
.
.
135         3           0           0             4
136         3           0           0             2
137         1           0           2        129600
138         3           0           0             4
.
.
.
```

The columns "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 pairs of haplotypes 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.

6 Haplotype Frequency Estimation by "haplo.em"

6.1 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 pairs of haplotypes for each subject are also computed. Unlike the usual EM which attempts to enumerate all possible pairs of haplotypes 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 sofware "snphap" by David Clayton. (http://www-gene.cimr.cam.ac.uk/clayton/software/). 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.

6.2 Example usage

This example uses the geno matrix for the 3 loci defined above, which contains the two subjects with some missing alleles.

```
> save.em <- haplo.em(geno=geno, locus.label=label, miss.val=c(0,NA))
```

Note on missing genotypes

Because of the missing data, the number of possible pairs of haplotypes is quite large, which increases the computation time. With the two subjects with missing genotypes included, it took 308.4 seconds of CPU time to complete the computations. If these two subjects are excluded, it took 1.4 seconds of CPU time. It is a good idea to preview the data for missing values using the function summaryGeno. If there are just a few subjects with missing alleles, it may be worthwhile to exclude them.

Print Method

To view the results in save.em, type either save.em or print(save.em):

```
> print(save.em)
================================================================================

                                    Haplotypes

================================================================================
```

```
DQB DRB  B hap.freq
  1  21   1  8  0.00232
  2  21   2  7  0.00227
  3  21   2 18  0.00227
  4  21   3  8  0.10408
  5  21   3 18  0.00229
  6  21   3 35  0.00568
  7  21   3 44  0.00381
  8  21   3 45  0.00227
  9  21   3 49  0.00227
 10  21   3 57  0.00227
 11  21   3 70  0.00227
.
.
.
================================================================================

                                    Details

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


lnlike =  -1846.835788
lr stat for no LD =  595.469334 , df =  123 , p-val =  0
```

Explanation of Results

The haplotypes and their estimated frequencies are listed, as well as a few details. The "`lr stat for no LD`" is the likelihood ratio statistic contrasting the `lnlike` for the estimated haplotype frequencies versus the `lnlike` assuming that alleles from all loci are in linkage equilibrium. Trimming by the progressive insertion algorithm can invalidate the `lr stat` and the degrees of freedom (`df`) – see the help file for `haplo.em`.

Summary Method

The function `summary` is used to see the list of haplotypes per subject, and their posterior probabilities:

```
> summary(save.em)

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

             Subjects: Haplotype Codes and Posterior Probabilities

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

subj.id hap1code hap2code posterior
  1       1       56       77   1.00000
  2       2       13      141   0.15143
  3       2       17      136   0.84857
  4       3       25      166   1.00000
  5       4       13       37   1.00000
  6       5       53       93   1.00000
  7       6        4       17   1.00000
```

```
  8       7      176      136   1.00000
  9       8      135       51   1.00000
 10       9      136      154   0.11921
.
.
.
455     218      102        4   1.00000
456     219        4      125   1.00000
457     220       98      136   1.00000
================================================================================

                    Number of haplotype pairs: max vs used

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


         1  2 72 135
     1  18  0  0   0
     2  52  2  0   0
     4 116 30  0   0
  1800   0  0  1   0
129600   0  0  0   1
```

Explanation of Results

The first part of `summary` 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=T` option:

```
> summary(save.em, show.haplo=T)

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

             Subjects: Haplotype Codes and Posterior Probabilities

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

subj.id hap1.DQB hap1.DRB hap1.B hap2.DQB hap2.DRB hap2.B posterior
    X56       1       31      11      61       32       4      62   1.00000
    X13       2       21       7       7       62       2      44   0.15143
    X17       2       21       7      44       62       2       7   0.84857
    X25       3       31       1      27       63      13      62   1.00000
   X131       4       21       7       7       31       7      44   1.00000
    X53       5       31      11      51       42       8      55   1.00000
     X4       6       21       3       8       21       7      44   1.00000
   X176       7       64      13      63       62       2       7   1.00000
   X135       8       61      13      44       31      11      44   1.00000
   X136       9       62       2       7       63       2      35   0.11921
   X140       9       62       2      35       63       2       7   0.88079
    X98      10       51       1      35       51       1      27   1.00000
```

6.3 Control Parameters for "haplo.em"

An additional argument can be passed to haplo.em, called "control". This is a list of parameters that control the EM algorithm based on progressive insertion of loci. The default values are set up by a function called haplo.em.control (see the help file for this function for a complete description). Although the user can accept the default values, there are times when they may need to be adjusted. For example, for small sample sizes and many possible haplotypes, finding the global maximum of the log-likelihood can be difficult. The algorithm uses multiple attempts to maximize the log-likelihood, starting each attempt with random starting values. If the results from haplo.em, haplo.score, or haplo.glm change when rerunning the analyses, this may be due to different maximizations of the log-likelihood. To avoid this, the user can increase the number of attempts (n.try) to maximize the log-likelihood, increase the batch size (insert.batch.size), or decrease the trimming threshold for posterior probabilities (min.posterior). If the EM algorithm fails to converge, try increasing the maximum number of iterations (max.iter). 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.

The example below illustrates how to set the number of trys to 20, and maximum number of iterations to 1,000:

```
> save.em <- haplo.em(geno=geno, locus.label=label, miss.val=c(0, NA),
                        control=haplo.em.control(n.try = 20, max.iter = 1000) )
```

6.4 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, y.bin to create subsets, along with the geno matrix (see section 7.3 for creation of y.bin):

```
> group.bin <- haplo.group(y.bin, geno, locus.label=label, miss.val=0)

> print(group.bin)
```

```
------------------------------------------------------------------------------
                        Counts per Grouping Variable Value
------------------------------------------------------------------------------


   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 0.00178      NA
  3  21  13  8 0.00274 0.00459      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 0.00568 0.00647      NA
  8  21   3 44 0.00381 0.00333 0.00830
  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 0.00794
.
.
 28  31  11 35 0.01753 0.01982 0.01587
 29  31  11 37 0.00699 0.00661 0.00794
 30  31  11 38 0.00699 0.00661 0.00794
.
.
.
```

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. 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 occurs when the haplotypes do not occur in the subgroups.

7 Haplotype Score Tests by "haplo.score"

The function haplo.score is used to compute score statistics to test associations 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, allows for adjustment for non-genetic covariates, and optionally allows computation of permutation p-values (which may be needed for sparse data). Details on the background and theory of the score statistics can be found in Schaid et al. (Schaid et al. 2002).

7.1 Quantitative Trait Analysis

First, analyze the quantitative trait called resp. A quantitative trait is identified by the option trait.type="gaussian" (a reminder that a gaussian distribution is assumed for the distribution of the error terms). The other arguments, all set to default values, are defined in the help file, viewed by typing help(haplo.score). To make reproducible output from haplo.score, set the seed for the random number generator using set.seed().

```
> score.gaus <- haplo.score(resp, geno, trait.type="gaussian",
      +               skip.haplo=.005, locus.label=label, simulate=FALSE)
```

To view results, either type the name of the object, score.gaus, or print(score.gaus).

```
> print(score.gaus)

      ------------------------------------------------------------------------
                                Global Score Statistics

      ------------------------------------------------------------------------

      global-stat = 46.77557, df = 39, p-val = 0.18345

      ------------------------------------------------------------------------

                                Haplotype-specific Scores

      ------------------------------------------------------------------------

            DQB DRB  B Hap-Freq Hap-Score   p-val
       [1,] 21   3   8 0.10408  -2.39627  0.01656
       [2,] 21   7  13 0.01076  -2.30095  0.02139
       [3,] 31   4  44 0.02851  -2.24652  0.02467
       [4,] 63  13  60 0.00575  -1.75685  0.07894
       [5,] 62   2  35 0.00756  -1.2105   0.22609
       [6,] 31  11  27 0.00618  -1.04042  0.29815
       [7,] 51   1  44 0.01772  -1.01301  0.31106
       [8,] 63  13  44 0.01606  -0.84453  0.39837
       [9,] 33   7  57 0.00682  -0.58522  0.5584
      [10,] 63   2   7 0.01337  -0.51076  0.60952
      .
      .
      .
```

Explanation of Results

The section `Global Score Statistics` prints results for testing an overall association between haplotypes and the response. The `global-stat` has an asymptotic χ^2 distribution, with degrees of freedom `df` and p-value as indicated.

`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.
`p-val` is the asymptotic chi-square (1 df) p-value.

In our example, the option `simulate=F` in the function `haplo.score` implied no permutation p-values (permutation p-values are computed when `simulate=T`.) Note that this table is sorted according to `Hap-Score`.

7.2 Ordinal Trait Analysis

To create an ordinal trait, convert `resp.cat` (a factor with levels "low", "normal", "high") to numeric values, `y.ord` (with levels 1, 2, 3).

```
> y.ord <- as.numeric(resp.cat)
```

Now use the option `trait.type = "ordinal"`

```
> score.ord <- haplo.score(y.ord, geno, trait.type="ordinal",
      +                          offset = NA, x.adj = NA, skip.haplo=.005,
      +                          locus.label=label, miss.val=0, simulate=FALSE)
> print(score.ord)

      ------------------------------------------------------------------------
                                Global Score Statistics

      ------------------------------------------------------------------------

      global-stat = 62.44401, df = 39, p-val = 0.00996
      ------------------------------------------------------------------------

                                Haplotype-specific Scores

      ------------------------------------------------------------------------
            DQB DRB  B Hap-Freq Hap-Score   p-val
       [1,] 21   7  13 0.01076  -3.67027  0.00024
       [2,] 21   3   8 0.10408  -2.79242  0.00523
       [3,] 31   4  44 0.02851  -2.6188   0.00882
       [4,] 63  13  60 0.00575  -2.35861  0.01834
       [5,] 33   7  57 0.00682  -0.93375  0.35043
       [6,] 33   9  60 0.00682  -0.93375  0.35043
```

WARNING FOR ORDINAL TRAITS:

When analyzing an ordinal trait with adjustment for covariates (using the `x.adj` option), the software requires the libraries `Design` and `Hmisc`, distributed by Frank Harrell, Ph.D. (see Harrell, FE. Regression Modeling Strategies, Springer-Verlag, NY, 2001). If the user does not have these libraries 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 libraries. To check whether your local system has these libraries, type

```
> library()
```

and you should then be able to examine the list of libraries available to you.

7.3 Binary Trait Analysis

Because "low" responders are of primary interest, create a binary trait that has values of 1 when response is "low", and 0 otherwise.

```
> y.bin <-ifelse(y.ord==1,1,0)
```

Now use the option `trait.type = "binomial"` in the `haplo.score` routine

```
> score.bin <- haplo.score(y.bin, geno, trait.type="binomial",
      +                         offset = NA, x.adj = NA, skip.haplo=.005,
      +                         locus.label=label, miss.val=0, simulate=FALSE)

> print(score.bin)

      ------------------------------------------------------------------------
                                Global Score Statistics

      ------------------------------------------------------------------------


      global-stat = 61.93695, df = 39, p-val = 0.01114

      ------------------------------------------------------------------------
                                Haplotype-specific Scores

      ------------------------------------------------------------------------

            DQB DRB  B Hap-Freq Hap-Score   p-val
       [1,] 62   2   7 0.05088  -2.18962  0.02855
       [2,] 51   1  35 0.03018  -1.58419  0.11315
       [3,] 63  13   7 0.01655  -1.56008  0.11874
       [4,] 21   7   7 0.01418  -1.56     0.11876
       [5,] 64  13  35 0.00897  -1.27328  0.20292
       [6,] 63  13  62 0.00866  -1.14178  0.25355
       [7,] 32   8   7 0.00682  -1.10475  0.26927
       [8,] 64  13  63 0.00682  -1.10475  0.26927
      .
      .
```

7.4 Plots and Haplotype Labels

A convenient way to view results from `haplo.score` is a plot of the haplotype frequencies (`Hap-Freq`) versus the haplotype score statistics (`Hap-Score`).

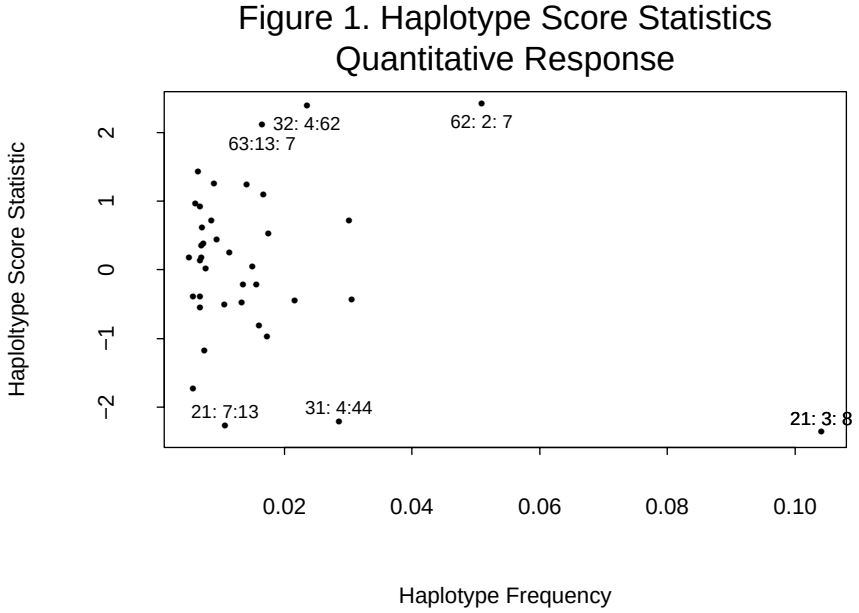
```
> plot(score.gaus)
> title("Figure 1. Haplotype Score Statistics\nQuantitative Response")
```

Some points on the plot may be of interest, perhaps due to their score statistic, or their haplotype frequency. To put haplotype labels on individual points in the plot, use the function `locator.haplo`. To use this feature, first type the following command:

```
> locator.haplo(score.gaus)
```

where `score.gaus` is the object where the results are stored.

Now, with the left mouse button, click on all the points of interest. After points are chosen, click on the middle mouse button. All the points are labeled with their haplotype labels, as illustrated in Figure 1.



If some of the haplotype labels overlap, it may be necessary to clean up the coordinates. To do this, save the x-y coordinates and haplotype text,

```
> loc.gaus <- locator.haplo(score.gaus)
```

and then fix some of the x-y coordinates in the `loc.gaus` list.

7.5 Skipping Rare Haplotypes

For the quantitative trait analyses, the option `skip.haplo=.005` was used to pool all haplotypes with frequencies < 0.005 into a common group. Now try a different cut-off, such as `skip.haplo=.01`.

```
> score.gaus.01 <- haplo.score(resp, geno, trait.type="gaussian",
      +                          offset = NA, x.adj = NA, skip.haplo=.01,
      +                          locus.label=label, miss.val=0, simulate=FALSE)

> print(score.gaus.01)

      ------------------------------------------------------------------------

                                Global Score Statistics

      ------------------------------------------------------------------------

      global-stat = 33.23181, df = 20, p-val = 0.03182

      ------------------------------------------------------------------------

                                Haplotype-specific Scores

      ------------------------------------------------------------------------

            DQB DRB  B Hap-Freq Hap-Score   p-val
       [1,] 21   3   8 0.10408  -2.39627  0.01656
       [2,] 21   7  13 0.01076  -2.30095  0.02139
       [3,] 31   4  44 0.02851  -2.24652  0.02467
       [4,] 51   1  44 0.01772  -1.01301  0.31106
       [5,] 63  13  44 0.01606  -0.84453  0.39837
       [6,] 63   2   7 0.01337  -0.51076  0.60952
       [7,] 31  11  44 0.01024  -0.49907  0.61773
       [8,] 32   4  60 0.03063  -0.46446  0.64232
       .
       .
       .
```

Note that by using a different value for `skip.haplo`, the global statistic and its p-value change (due to decreased `df`), but the haplotype-specific scores do not change.

7.6 Haplotype scores, adjusted for covariates

First set up a covariate matrix, with the first column for `male` (1 if male; 0 if female), and the second column for `age` (in months).

```
> x.ma <- cbind(male, age)
```

Now use this matrix as an argument to `haplo.score`.

```
> score.gaus.adj <- haplo.score(resp, geno, trait.type="gaussian",
      +                         offset = NA, x.adj = x.ma, skip.haplo=.005,
      +                         locus.label=label, miss.val=0, simulate=FALSE)

> print(score.gaus.adj)

      ------------------------------------------------------------------------
                               Global Score Statistics

      ------------------------------------------------------------------------


      global-stat = 46.99283, df = 39, p-val = 0.17773

      ------------------------------------------------------------------------
                              Haplotype-specific Scores

      ------------------------------------------------------------------------

            DQB DRB  B Hap-Freq Hap-Score   p-val
       [1,] 21   3   8 0.10408  -2.40966  0.01597
       [2,] 21   7  13 0.01076  -2.29137  0.02194
       [3,] 31   4  44 0.02851  -2.25648  0.02404
       [4,] 63  13  60 0.00575  -1.77443  0.07599
       [5,] 62   2  35 0.00756  -1.21505  0.22435
       [6,] 31  11  27 0.00618  -1.03577  0.30031
       [7,] 51   1  44 0.01772  -1.00684  0.31401
       [8,] 63  13  44 0.01606  -0.83952  0.40118
```

When adjusting for covariates, all score statistics can change, although not by much in this example.

7.7 Permutation p-values

Permutation p-values are computed when `simulate=TRUE.` In addition to a global statistic, and haplotype-specific statistics, a "`max-stat`" statistic, and corresponding permutation p-value, is computed. The `max-stat` is the maximum among all haplotype-specific score statistics. Because the distribution of this statistic is unknown, the p-value for `max-stat` is given only when permutations are requested. If only a few haplotypes are associated with the trait, the `max-stat` should have greater power than the global statistic.

The `score.sim.control` function manages simulation control parameters. Simulated statistics are based on randomly permuting the trait and covariates (same order for both), but not the geno matrix, and then computing the haplotype score statistics, adjusted for covariates. `haplo.score` employs the simulation p-value precision criteria of Besag and Clifford (Besag and Clifford 1991). These criteria ensure that the permutation 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 for individual haplotypes. Permutations beyond this minimum are then conducted until the sample standard errors for simulated p-values for both the global and max score statistics are less than a threshold (`p.threshold * p-value`). The default value for `p.threshold`=¼ provides a two-sided 95% confidence interval for the p-value with a width that is approximately as wide as the p-value itself. Simulations are more precise for smaller p-values.

The following example illustrates computation of permutation p-values with `min.sim=1000`.

```
> score.bin.sim <- haplo.score(y.bin, geno, trait.type="binomial",
      +                         offset = NA, x.adj = NA, skip.haplo=.005,
      +                         locus.label=label, miss.val=0, simulate=TRUE,
      +                         sim.control=score.sim.control())

> print(score.bin.sim)

      ------------------------------------------------------------------------

                                 Global Score Statistics

      ------------------------------------------------------------------------

      global-stat = 61.93695, df = 39, p-val = 0.01114

      ------------------------------------------------------------------------
      ------
                            Global Simulation p-value Results

      ------------------------------------------------------------------------
      ------

      Global sim. p-val =  0.01043
      Max-Stat sim. p-val =  0.00758
      Number of Simulations, Global:  2110 , Max-Stat: 2110

      ------------------------------------------------------------------------
                                Haplotype-specific Scores

      ------------------------------------------------------------------------

            DQB DRB  B Hap-Freq Hap-Score   p-val sim p-val
       [1,] 62   2   7 0.05088  -2.18962  0.02855 0.02512
       [2,] 51   1  35 0.03018  -1.58419  0.11315 0.13839
       [3,] 63  13   7 0.01655  -1.56008  0.11874 0.20332
       [4,] 21   7   7 0.01418  -1.56     0.11876 0.14692
       [5,] 64  13  35 0.00897  -1.27328  0.20292 0.32701
       [6,] 63  13  62 0.00866  -1.14178  0.25355 0.36872
      .
      .
      .
```

7.8 Combine Score and Group Results by "haplo.score.merge"

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

```
> merge.bin <- haplo.score.merge(score.bin, group.bin)

> print(merge.bin)

      ------------------------------------------------------------------------

                    Haplotype Scores, p-values, and Frequencies By Group

      ------------------------------------------------------------------------

         DQB DRB  B Hap.Score   p.val Hap.Freq y.bin.0 y.bin.1
       1  62   2  7  -2.18962 0.02855  0.05088 0.06789 0.01587
       2  51   1 35  -1.58419 0.11315  0.03018 0.03755 0.00893
       3  63  13  7  -1.56008 0.11874  0.01655 0.02182      NA
       4  21   7  7  -1.56000 0.11876  0.01418 0.01969      NA
       5  64  13 35  -1.27328 0.20292  0.00897 0.01310      NA
       6  63  13 62  -1.14178 0.25355  0.00866 0.01274      NA
       7  32   8  7  -1.10475 0.26927  0.00682 0.00955      NA
       8  64  13 63  -1.10475 0.26927  0.00682 0.00955      NA
      .
      .
      .
```

Explanation of Results

The first column is a row index, the next columns (3 in this example) illustrate the haplotype, the column "Hap.Score" is the score statistic and "p.val" the corresponding chi-square p-value. Hap.prob is the haplotype frequency for the total sample, and the remaining last two columns are the estimated haplotype frequencies for each of the group levels (y.bin in this example). The default print method only prints results for haplotypes appearing in the haplo.score output. To view all haplotypes, use the print option all.haps=T, which prints records for all haplotypes from the haplo.group output. The output is ordered by the score statistic, but the order.by parameter can specify ordering by haplotypes or by haplotype frequency. See the help file for print.haplo.score.merge for details on printing options.

7.9 Apply Score Tests to Sub-haplotypes by "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.

```
> geno.11 <- hla.demo[,-c(1:4)]
> label.11 <- c("DPB","DPA","DMA","DMB","TAP1","TAP2","DQB","DQA","DRB","B","A")
```

Now 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 same gaussian trait as above.

```
> score.slide.gaus <- haplo.score.slide(resp, geno.11, trait.type="gaussian",
                            n.slide=3, skip.haplo=.005, locus.label=label.11)

> print(score.slide.gaus)

        start.locus score.global.p score.global.p.sim score.max.p.sim
      1           1      0.2963692                 NA              NA
      2           2      0.0078741                 NA              NA
      3           3      0.2270879                 NA              NA
      4           4      0.7663349                 NA              NA
      5           5      0.2172153                 NA              NA
      6           6      0.2111512                 NA              NA
      7           7      0.2178189                 NA              NA
      8           8      0.0395811                 NA              NA
      9           9      0.0300852                 NA              NA
```

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. The last two columns are the simulated p-values for the global and maximum score statistics, respectively. If `simulate=T` was specified in the function call, the simulated p-values would be present.

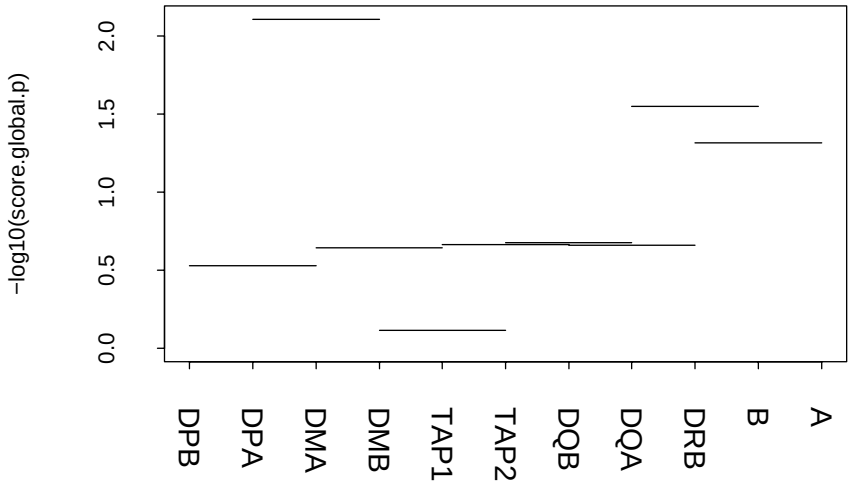
Plot Results from "`haplo.score.slide`"

The results from `haplo.score.slide` can be easily viewed in a plot, by the command

```
> plot(score.slide.gaus,cex=.6)
> title("Figure 2. Global p-values for\n sub-haplotypes; Gaussian
        Response",cex=.6)
```

with results illustrated in Figure 2.

Figure 2. Global p−values for
sub−haplotypes; Gaussian Response



The x-axis has tick marks for each locus, and the y-axis is the -log10(p-value). 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 -log10(p-value) across the loci over which it was calculated. For example, the p-value `score.global.p = 0.0078741` for loci 2-4 will plot as a horizontal line plotted at y=2.1 covering the 2nd, 3rd, and 4th x-axis tick marks.

8 Regression Models with "`haplo.glm`"

The function `haplo.glm` computes the regression of a trait on haplotypes, and possibly other covariates and their interactions with haplotypes. Although this function is based on a generalized linear model, only two types of traits are currently supported: 1) quantitative traits with a normal (gaussian) distribution and identity link, and 2) binomial traits with a logit-link function. 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 posterior probabilities. See Lake (Lake et al. 2003) for details.

8.1 Setting up the `data.frame`

A critical distinction between `haplo.glm` and all other functions in Haplo Stats is that the definition of the regression model follows the S-PLUS/R formula standard. So, a `data.frame` must be defined, and this `data.frame` must contain the trait, a special kind of genotype matrix (called `geno` in this example) that contains the genotypes of the marker loci, and possibly other covariates and weights for the subjects. The key feature of this `data.frame` is how `geno` is created. To account for character, numeric, or factor alleles, and to keep the columns of `geno` as a single unit when inserting into, and extracting from, a `data.frame` (and not separate columns of the `data.frame`), the `geno` matrix is created by the function `setupGeno`. This function recodes alleles to integer values (the allele codes are an attribute of the returned object), and returns a `model.matrix`, which can then be inserted into a `data.frame`.

The following steps are necessary to convert `geno` to the appropriate `model.matrix` with allele labels usable in `haplo.glm`.

```
> geno <- as.matrix(hla.demo[,c(17,18,21:24)])
> geno <- setupGeno(geno, miss.val=c(0,NA))
```

Note that `geno` has `'unique.alleles'` as an attribute (a list of vectors, where a vector contains the allele labels for a locus). This list, `'unique.alleles'`, must be passed to `haplo.glm` so the original allele labels can be used to label haplotypes.

Now create a `data.frame` for the regressions

```
my.data <- data.frame(geno, age=hla.demo$age, male=hla.demo$male,
y=hla.demo$resp, y.bin=y.bin)
```

8.2 Regression for a Quantitative Trait

The following illustrates how to fit a regression of quantitative trait `y` on the haplotypes defined by the matrix `geno`, and the covariate `male`. The control parameter, `haplo.freq.min`, is discussed below under the heading “Explanation of Results”, as well as in section 8.5.

```
> fit.gaus <- haplo.glm(y ~ male + geno, family = gaussian,
     na.action="na.geno.keep", locus.label=label,
     allele.lev=attributes(geno)$unique.alleles, data=my.data
     control = haplo.glm.control(haplo.freq.min=0.02))
```

To view the results of a model fit, type either the name of the saved object (e.g., `fit.gaus`), or use `print`:

```
> print(fit.gaus)
```

```
> fit.gaus
Call:
```

```
haplo.glm(formula = y ~ male + geno, family = gaussian, data = my.data,
      na.action = "na.geno.keep", locus.label = locus.label, allele.lev =
      attributes(geno)$unique.alleles, control =
      haplo.glm.control(haplo.freq.min = 0.02))

Coefficients:
coef    se t.stat    pval
(Intercept)  1.0652 0.343  3.107 0.00215
       male  0.0975 0.155  0.629 0.52989
    geno.17  0.2659 0.456  0.584 0.56002
    geno.33 -0.3189 0.343 -0.928 0.35425
    geno.76  0.2228 0.361  0.617 0.53775
    geno.77  1.1407 0.384  2.972 0.00330
    geno.98  0.5546 0.364  1.523 0.12933
   geno.136  0.9841 0.304  3.242 0.00138
  geno.rare  0.3971 0.182  2.184 0.03010

Haplotypes:
           DQB DRB  B hap.freq
   geno.17  21   7 44   0.0211
   geno.33  31   4 44   0.0286
   geno.76  32   4 60   0.0303
   geno.77  32   4 62   0.0239
   geno.98  51   1 35   0.0301
  geno.136  62   2  7   0.0502
 geno.rare   *   *  *   0.7118
haplo.base  21   3  8   0.1041
```

Explanation of Results

The above table for `Coefficients` lists the estimated value (`coef`), its standard error (`se`), the corresponding t-statistic (`t.stat`) , and p-value (`pval`). The labels for haplotype coefficients are a paste of the matrix defining the genotypes (`geno` in the above example) and the haplotype numbers. The haplotypes corresponding to these haplotype numbers are listed in the above table under `Haplotypes`, along with the estimates of the haplotype frequencies (`hap.freq`). The rare haplotypes (those with frequencies less than `haplo.freq.min`, 0.02 in the above example) are pooled into a single category labeled `rare`. The haplotype chosen as the base-line category for the design matrix (most frequent haplotype is the default) is labeled as `haplo.base`.

8.3 Fitting Haplotype x Covariate Interactions

Interactions are fit by the usual model syntax, using a "*" to indicate main effects and interactions,

```
> fit.inter <- haplo.glm(y ~ male * geno, family = gaussian,
                         data=my.data, locus.label= label,
                         allele.lev=attributes(geno)$unique.alleles
                         control = haplo.glm.control(haplo.freq.min
                         = 0.02))

> fit.inter
Call:
haplo.glm(formula = y ~ male * geno, family = gaussian, data = my.data,
```

```
      na.action = "na.geno.keep", locus.label = label, allele.lev =
      attributes(geno)$unique.alleles, control =
      haplo.glm.control(haplo.freq.min = 0.02))

Coefficients:
coef    se  t.stat    pval
  (Intercept)  0.9800 0.303  3.2344 0.00142
         male  0.2548 0.314  0.8107 0.41850
      geno.17  0.1018 0.465  0.2190 0.82689
      geno.33 -0.1746 0.596 -0.2932 0.76967
      geno.76  0.8033 0.590  1.3611 0.17498
      geno.77  0.4939 0.489  1.0103 0.31355
      geno.98  0.5203 0.400  1.3005 0.19489
     geno.136  1.1604 0.354  3.2787 0.00123
    geno.rare  0.4525 0.182  2.4802 0.01394
  malegeno.17  0.5444 0.720  0.7564 0.45027
  malegeno.33 -0.2794 0.668 -0.4185 0.67605
  malegeno.76 -0.8893 0.697 -1.2752 0.20368
  malegeno.77  1.2656 0.655  1.9317 0.05478
  malegeno.98  0.0536 0.661  0.0811 0.93547
 malegeno.136 -0.4488 0.521 -0.8614 0.39004
malegeno.rare -0.0971 0.206 -0.4701 0.63878

Haplotypes:
           DQB DRB  B hap.freq
   geno.17  21   7 44   0.0217
   geno.33  31   4 44   0.0285
   geno.76  32   4 60   0.0301
   geno.77  32   4 62   0.0241
   geno.98  51   1 35   0.0301
  geno.136  62   2  7   0.0504
 geno.rare   *   *  *   0.7110
haplo.base  21   3  8   0.1041
```

Explanation of Results

The listed results are as explained under section 8.2. The only difference is that the interaction coefficients are labeled as a paste of the covariate (`male` in this example) and the name of the haplotype.

8.4 Regression for a Binomial Trait

The following illustrates the fitting of a binomial trait:

```
> fit.bin <- haplo.glm(y.bin ~ male + geno, family = binomial,
                       data=my.data, locus.label= label,
                       allele.lev=attributes(geno)$unique.alleles,
                       x=T, control = haplo.glm.control(haplo.freq.min
                       = 0.02))
> print(fit.bin)
Call:
haplo.glm(formula = y.bin ~ male + geno, family = binomial, data = my.data,
      na.action = "na.geno.keep", locus.label = locus.label,
      allele.lev=attributes(geno)$unique.alleles, control =
      haplo.glm.control(haplo.freq.min = 0.02), x = T)
```

```
Coefficients:
              coef    se t.stat     pval
(Intercept)  1.540 0.420  3.670 3.07e-04
       male -0.480 0.324 -1.482 1.40e-01
    geno.17 -0.587 0.707 -0.831 4.07e-01
    geno.33  0.373 0.624  0.598 5.50e-01
    geno.76 -0.999 0.689 -1.450 1.48e-01
    geno.77 -1.404 0.773 -1.815 7.09e-02
    geno.98 -2.583 0.710 -3.641 3.42e-04
   geno.136 -2.717 0.759 -3.579 4.28e-04
  geno.rare -1.261 0.253 -4.980 1.32e-06

Haplotypes:
           DQB DRB  B hap.freq
   geno.17  21   7 44   0.0214
   geno.33  31   4 44   0.0284
   geno.76  32   4 60   0.0306
   geno.77  32   4 62   0.0235
   geno.98  51   1 35   0.0298
  geno.136  62   2  7   0.0517
 geno.rare   *   *  *   0.7105
haplo.base  21   3  8   0.1040
```

Interpreting Results for a Binomial Trait

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 incidence cases), haplotypes associated with disease will be over-represented in the case-control 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 (Stram et al. 2003) for background on using weights, and see the help file for `haplo.glm` for how to implement weights with `haplo.glm`.

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 (Stram et al. 2003) and Epstein (Epstein and Satten 2003) for further details.

8.5 Control Parameters and Genetic Models

A key parameter for `haplo.glm` is `control`, which is a list of parameters that control the procedures of `haplo.glm`. This control list is set up by the function `haplo.glm.control`. A parameter of this control function is `haplo.effect`, which instructs whether the haplotype effects are fit as additive, dominant, or recessive. That is, `haplo.effect` determines whether the

covariate (x) coding of haplotypes is "additive" (causing x = 0, 1, or 2, the count of a particular haplotype), "dominant" (causing x = 1 if heterozygous or homozygous carrier of a particular haplotype; x = 0 otherwise), or "recessive" (causing x = 1 if homozygous for a particular haplotype; x = 0 otherwise).

```
> fit.dom <- haplo.glm(y ~ male + geno, family = gaussian,
                 data=my.data, locus.label=label,
                 allele.lev=attributes(geno)$unique.alleles,
                 control = haplo.glm.control(haplo.effect = "dom",
                 haplo.freq.min = 0.02) )
```

The other control parameter used above, `haplo.freq.min`, is the minimum haplotype frequency required for a haplotype to be included in the regression model as its own effect. See the help file for `haplo.glm.control` for further control parameters.

9 License and Warranty

License:

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.

11 Appendix: 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 i^{th} 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 heterozygous or homozygous.

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 $a_i(a_i - 1)/2$ possible genotypes. A function to perform this counting for heterozygous loci is denoted $g(a_i)$. These functions and counts are summarized in Table 1.

Table 1. Number of possible genotypes at a locus with missing alleles according to whether the genotype is homozygous or heterozygous.

Number of missing alleles	Homozygous function, $f(a_i)$	Heterozygous function, $g(a_i)$
1	1	$a_i - 1$
2	a_i	$a_i(a_i - 1)/2$

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

$$a_i\, 2^x \; + [a_i (a_i - 1)/2]\; 2^{x+1},$$

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 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 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 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

$$\text{Total Pairs} = \sum_{i=0}^{m} \sum_{combo} C(combo, i)$$

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 second and third as homozygous), and $C(combo, i)$ is the corresponding count for this pattern when there are *i* loci that are heterozygous (e.g., for *m* = 3 and *i* = 1, $g(a_1) f(a_2) f(a_3) 2^{x+1}$, as illustrated in Table 2).

Table 2. Summands to count the number of pairs of haplotypes consistent with observed marker phenotypes when there are *m* loci with missing alleles.

	Number of loci with missing alleles that are considered as heterozygous			
m	0	1	2	3
0	2^x			
1	$f(a_1) 2^x$	$g(a_1) 2^{x+1}$		
2	$f(a_1) f(a_2) 2^x$	$g(a_1) f(a_2) 2^{x+1}$ $f(a_1) g(a_2) 2^{x+1}$	$g(a_1) g(a_2) 2^{x+2}$	
3	$f(a_1) f(a_2) f(a_3) 2^x$	$g(a_1) f(a_2) f(a_3) 2^{x+1}$ $f(a_1) g(a_2) f(a_3) 2^{x+1}$ $f(a_1) f(a_2) g(a_3) 2^{x+1}$	$g(a_1) g(a_2) f(a_3) 2^{x+2}$ $g(a_1) f(a_2) g(a_3) 2^{x+2}$ $f(a_1) g(a_2) g(a_3) 2^{x+2}$	$g(a_1) g(a_2) g(a_3) 2^{x+3}$

12 References

Besag J, Clifford P (1991) Sequential Monte Carlo p-Values. Biometrika 78:301-304

Epstein M, Satten G (2003) Inference on haplotype effects in case-control studies using unphased genotype data. Submitted

Lake S, Lyon H, Silverman E, Weiss S, Laird N, Schaid D (2003) Estimation and tests of haplotype-environment interaction when linkage phase is ambiguous. Human Heredity 55:56-65

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA (2002) Score tests for association between traits and haplotypes when linkage phase is ambiguous. Am J Hum Genet 70:425-34

Stram D, Pearce C, Bretsky P, Freedman M, Hirschhorn J, Altshuler D, Kolonel L, Henderson B, Thomas D (2003) Modeling and E-M estimation of haplotype-specific relative risks from genotype data for case-control study of unrelated individuals. Hum Hered 55:179-190