Haplo Stats
(version 1.4.0)

Statistical Methods for Haplotypes When Linkage Phase is Ambiguous

Jason P. Sinnwell and Daniel J. Schaid
Mayo Clinic, Rochester MN USA

October 3, 2008

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), while also allowing for missing alleles.

The primary 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:* 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.
- *seqhap:* build a haplotype model by adding markers sequentially until no improvement in the model

This version of *haplo.stats* contains the addition of power and sample size functions, and some updates to existing functions. Updates are noted in section 4.4 for control parameters in *haplo.em*, section 6.7 for the *eps.svd* parameter in *haplo.score*, section 7.1 regarding parameter changes in *haplo.glm*, section 7.6.3 for discussion on rare haplotypes in *haplo.glm*, section 8.2 for a parameter change in *haplo.cc*, and in section 8.5 for permutation parameter changes in *seqhap*. The newly added functions are described in section 5, and a general description is as follows:

- *haplo.power.qt:* power and sample size calculations for quantitative trait haplotype associations
- *haplo.power.cc:* power and sample size calculations for case-control haplotype associations

2 Operating System and Installation

Haplo Stats version 1.4.0 is written for both S-PLUS (version 8.0.1) and R (version 2.7.1) for Unix. It has been placed on the Comprehensive R Archive Network (CRAN), and is made available on other systems through CRAN. Installation procedures for S-PLUS and R systems will vary; the Unix installations are explained in the *README.haplo.stats* text file, located at the top level of the *haplo.stats* directory. The procedures for running analyses are the same for S-PLUS and R, following instructions in this document.

3 Getting Started

After installing the Haplo Stats package, the routines are available by starting an S-PLUS or R session and loading the package. If *haplo.stats* is installed for global use, load the library as done below. If installed as a local library, specify its location in the *lib.loc* parameter as shown in comments(##).

```
> library(haplo.stats)

  ## if local library, use:
          ## library(haplo.stats, lib.loc="/local/install/path/")
```

For users who are new to the S-PLUS or R environments, note the following basic concepts. In the following examples, a user enters the indented text following the prompt ">", and the output results follow. Later, when executing a function in the session, the general syntax will appear like *'myresult <- myfunction(x)'* where the results of *myfunction*, operating on *x*, are saved in *myresult*. A user may view the contents of *myresult*, or make use of the contents in a calculation. The examples provided in this document provide enough tools for a typical haplotype analysis. More information can be found within the function help files, viewed here for *haplo.em*.

```
> help(haplo.em)
```

3.1 Example Data

The *haplo.stats* library contains three example data sets, one of which 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()*. The *hla.demo* data is already available in S-PLUS. Below, 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.

```
> data(hla.demo)
> names(hla.demo)

  [1] "resp"     "resp.cat" "male"     "age"      "DPB.a1"
  [6] "DPB.a2"   "DPA.a1"   "DPA.a2"   "DMA.a1"   "DMA.a2"
 [11] "DMB.a1"   "DMB.a2"   "TAP1.a1"  "TAP1.a2"  "TAP2.a1"
 [16] "TAP2.a2"  "DQB.a1"   "DQB.a2"   "DQA.a1"   "DQA.a2"
 [21] "DRB.a1"   "DRB.a2"   "B.a1"     "B.a2"     "A.a1"
 [26] "A.a2"

> attach(hla.demo)

         The following object(s) are masked from hla.demo ( position 3 ) :

          A.a1 A.a2 B.a1 B.a2 DMA.a1 DMA.a2 DMB.a1 DMB.a2 DPA.a1 DPA.a2 DPB.a1
```

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.

3.2 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 $2K$. 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.

```
> 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.

3.3 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.

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

```
    missing0 missing1 missing2 N.enum.rows
1          3        0        0           4
2          3        0        0           4
3          3        0        0           4
4          3        0        0           2
5          3        0        0           4
6          3        0        0           2
7          3        0        0           4
8          3        0        0           2
9          3        0        0           2
10         3        0        0           1
80         3        0        0           4
81         2        0        1        1800
82         3        0        0           2
83         3        0        0           1
84         3        0        0           2
85         3        0        0           4
135        3        0        0           4
136        3        0        0           2
137        1        0        2      129600
138        3        0        0           4
139        3        0        0           4
140        3        0        0           4
```

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 overwhelming.

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* summarizes 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.

```
> table(geno.desc[, 3])

  0   1   2 
218   1   1 

> miss.all <- which(geno.desc[, 3] == 3)
> hla.demo.updated <- hla.demo[-miss.all, ]
```

3.4 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*). Random numbers can be controlled when trying to reproduce calculations involving random numbers. Random numbers in S-PLUS and R are controlled by 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 (i.e., *haplo.em*, *haplo.score*, *haplo.score.slide*, *haplo.glm*, *haplo.group*, *haplo.cc*) to make results reproducible . Section 7.6.2 shows one example of setting the seed for *haplo.glm*. We illustrate setting the seed below.

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

4 Haplotype Frequency Estimation: *haplo.em*

4.1 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[4]. 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.

4.2 Example Usage

Use *haplo.em* on *geno* for the 3 loci defined above, then view the results stored in *save.em*. In this example we show just 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 the print methods in the Haplo Stats package to shorten the lengthy results for this user guide. In practice, it is best to exclude this parameter so that the default will print all results.)

```
> save.em <- haplo.em(geno = geno, locus.label = label,
+     miss.val = c(0, NA))
> print(save.em, nlines = 10)
```

```
=============================================================
                          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  0.00570
7   21   3 44  0.00378
8   21   3 45  0.00227
9   21   3 49  0.00227
10  21   3 57  0.00227
=============================================================
                           Details
=============================================================
lnlike =  -1847.675
lr stat for no LD =  632.5085 , df =  125 , 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*).

4.3 Summary Method

The *summary* on *save.em* shows the list of haplotypes per subject, and their posterior probabilities:

```
> summary(save.em, nlines = 7)
```

```
============================================================
          Subjects: Haplotype Codes and Posterior
                        Probabilities
============================================================
  subj.id hap1code hap2code posterior
1       1       78       58   1.00000
2       2      143       13   0.12532
3       2      138       17   0.87468
4       3      168       25   1.00000
5       4       13       39   0.28621
6       4       17       38   0.71379
7       5       94       55   1.00000
============================================================
           Number of haplotype pairs: max vs used
============================================================

x           1   2   3  84 135
  1        18   0   0   0   0
  2        50   4   0   0   0
  4       116  29   1   0   0
  1800      0   0   0   1   0
  129600    0   0   0   0   1
```

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:

```
> summary(save.em, show.haplo = TRUE, nlines = 7)

  ============================================================
           Subjects: Haplotype Codes and Posterior
                        Probabilities
  ============================================================
      subj.id hap1.DQB hap1.DRB hap1.B hap2.DQB hap2.DRB
  78        1       32        4     62       31       11
  143       2       62        2     44       21        7
  138       2       62        2      7       21        7
  168       3       63       13     62       31        1
  13        4       21        7      7       31        7
  17        4       21        7     44       31        7
  94        5       42        8     55       31       11
      hap2.B posterior
  78      61   1.00000
  143      7   0.12532
  138     44   0.87468
  168     27   1.00000
  13      44   0.28621
  17       7   0.71379
  94      51   1.00000
  ============================================================
            Number of haplotype pairs: max vs used
  ============================================================

  x          1   2   3  84 135
    1       18   0   0   0   0
    2       50   4   0   0   0
    4      116  29   1   0   0
    1800     0   0   0   1   0
    129600   0   0   0   0   1
```

4.4 Control Parameters for *haplo.em (UPDATED)*

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 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 results 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 illustrates the syntax for increasing the number of tries to 20, and the batch size to 2, since not much more can be done for three markers.

```
> save.em <- haplo.em(geno = geno, locus.label = label,
+     miss.val = c(0, NA), control = haplo.em.control(n.try = 20,
+         insert.batch.size = 2))
```

4.5 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.

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

5 Power and Sample Size for Haplotype Association Studies (*NEW*)

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[11], 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.

5.1 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[10] and an F-test[15]. 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 of 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.

```
> data(hapPower.demo)
> hmat <- hapPower.demo[, -6]
> hfreq <- hapPower.demo[, 6]
> hrisk <- which(hmat$loc.2 == 1 & hmat$loc.3 ==
+     1)
> hbase <- 1
> hbeta.list <- find.haplo.beta.qt(haplo = hmat,
+     haplo.freq = hfreq, base.index = hbase, haplo.risk = hrisk,
+     r2 = 0.01, y.mu = 0, y.var = 1)
> hbeta.list

 $r2
 [1] 0.01

 $beta
  [1] -0.03892497  0.00000000  0.00000000  0.00000000
  [5]  0.00000000  0.00000000  0.00000000  0.27636731
  [9]  0.00000000  0.27636731  0.00000000  0.00000000
 [13]  0.00000000  0.00000000  0.00000000  0.27636731
 [17]  0.27636731  0.00000000  0.00000000  0.00000000
 [21]  0.00000000

 $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 = 0.05, power = 0.8)
> ss.qt

 $ss.phased.haplo
 [1] 2091

 $ss.unphased.haplo
```

```
[1] 2826

$power.phased.haplo
[1] 0.8

$power.unphased.haplo
[1] 0.8
```

```
> power.qt <- haplo.power.qt(hmat, hfreq, hbase,
+     hbeta.list$beta, y.mu = 0, y.var = 1, alpha = 0.05,
+     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
```

5.2 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 5.1, give the risk haplotypes an odds ratio of 1.50, and specify a population disease prevalance 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 $(4,566)$.

```
> cc.OR <- 1.5
> hbeta.cc <- numeric(length(hfreq))
> hbeta.cc[hrisk] <- log(cc.OR)
> ss.cc <- haplo.power.cc(hmat, hfreq, hbase, hbeta.cc,
+     case.frac = 0.5, prevalence = 0.1, alpha = 0.05,
```

```
+     power = 0.8)
> ss.cc

 $ss.phased.haplo
 [1] 3454

 $ss.unphased.haplo
 [1] 4566

 $power.phased.haplo
 [1] 0.8

 $power.unphased.haplo
 [1] 0.8

> power.cc <- haplo.power.cc(hmat, hfreq, hbase,
+     hbeta.cc, case.frac = 0.5, prevalence = 0.1,
+     alpha = 0.05, sample.size = 4566)
> power.cc

 $ss.phased.haplo
 [1] 4566

 $ss.unphased.haplo
 [1] 4566

 $power.phased.haplo
 [1] 0.9206568

 $power.unphased.haplo
 [1] 0.8000695
```

6 Haplotype Score Tests: *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 and allows for adjustment for non-genetic covariates. A new feature is that 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.[10].

6.1 Quantitative Trait Analysis

First, 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 distribution of 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 6.6. Below is an example of running *haplo.score* with a quantitative trait, then viewing the results using the *print* method (again, output shortened by *nlines*).

```
> 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
--------------------------------------------------------------

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

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 χ^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 χ^2_1 p-value, calculated from the square of the score statistic.

6.2 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 *resp.cat* is "low", and 0 otherwise. Then in *haplo.score* specify the parameter *trait.type="binomial"*.

```
> 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)
```

```
------------------------------------------------------------
              Haplotype Effect Model: additive
------------------------------------------------------------
------------------------------------------------------------
                  Global Score Statistics
------------------------------------------------------------

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

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

      DQB DRB B  Hap-Freq Hap-Score p-val
 [1,] 62  2   7  0.05098  -2.19387  0.02824
```

```
 [2,] 51  1   35 0.03018  -1.58421  0.11315
 [3,] 63  13  7  0.01655  -1.56008  0.11874
 [4,] 21  7   7  0.01246  -1.47495  0.14023
 [5,] 32  4   7  0.01678  -1.00091  0.31687
 [6,] 32  4   62 0.02349  -0.6799   0.49657
 [7,] 51  1   27 0.01505  -0.66509  0.50599
 [8,] 31  11  35 0.01754  -0.5838   0.55936
 [9,] 31  11  51 0.01137  -0.43721  0.66196
[10,] 51  1   44 0.01731  0.00826   0.99341
```

6.3 Ordinal Trait Analysis

To create an ordinal trait, here we convert *resp.cat* (described above) to numeric values, *y.ord* (with levels 1, 2, 3). For *haplo.score*, use *y.ord* as the response variable, and set the parameter *trait.type* = ”ordinal”.

```
> 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
--------------------------------------------------------------

     DQB DRB B  Hap-Freq Hap-Score p-val
[1,] 32  4   62 0.02349  -2.17133  0.02991
[2,] 21  3   8  0.10408  -1.34661  0.17811
[3,] 32  4   7  0.01678  -1.09487  0.27357
[4,] 62  2   7  0.05098  -0.96874  0.33268
[5,] 21  7   44 0.02332  -0.83747  0.40233
```

```
[6,] 63  13  7  0.01655  -0.80787  0.41917
[7,] 21  7   7  0.01246  -0.63316  0.52663
```

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 [6]. 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. Check the list of your local libraries in the list shown from entering *library()* in your prompt.

6.4 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, though not by much in this example.

```
> 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
------------------------------------------------------------

      DQB DRB B  Hap-Freq Hap-Score p-val
 [1,] 21  3   8  0.10408  -2.4097   0.01597
 [2,] 31  4   44 0.02849  -2.25293  0.02426
```

```
 [3,] 51  1   44 0.01731  -0.98763  0.32333
 [4,] 63  13  44 0.01606  -0.83952  0.40118
 [5,] 63  2   7  0.01333  -0.48483  0.6278
 [6,] 32  4   60 0.0306   -0.46476  0.64211
 [7,] 21  7   44 0.02332  -0.41249  0.67998
 [8,] 62  2   44 0.01367  -0.26443  0.79145
 [9,] 62  2   18 0.01545  -0.20425  0.83816
[10,] 51  1   27 0.01505  0.02243   0.9821
```

6.5 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*). 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 *locator.haplo(score.gaus)*, which is similar to *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 *locator.haplo* function.

```
> plot(score.gaus.add)
> pts.haplo <- list(x.coord = c(0.05098, 0.03018,
+     0.1), y.coord = c(2.1582, 0.45725, -2.1566),
+     hap.txt = c("62:2:7", "51:1:35", "21:3:8"))
> text(x = pts.haplo$x.coord, y = pts.haplo$y.coord,
+     labels = pts.haplo$hap.txt)
```

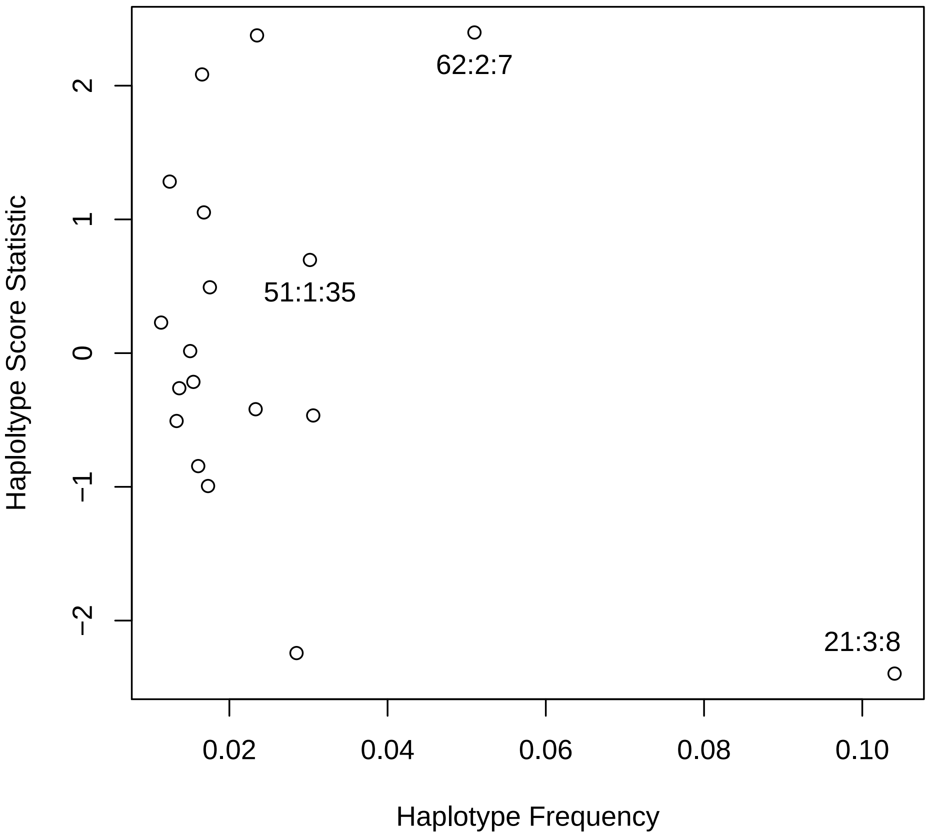

Figure 1: Haplotype Statistics: Score vs. Frequency, Quantitative Response

6.6 Skipping Rare Haplotypes

For the *haplo.score*, the *skip.haplo* and *min.count* parameters control which rare haplotypes are pooled into a common group. The *min.count* parameter is a recent addition to *haplo.score*, yet it does the same task as *skip.haplo* and is the same idea as *haplo.min.count* used in *haplo.glm.control* for *haplo.glm*. As a guideline, you may wish to set *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 $count = 2 \times N \times f$. Alternatively, you can choose $skip.haplo = \frac{count}{2 \times N}$. In the following example we try a different cut-off than before, *min.count=10*, which corresponds to *skip.haplo* of $10 \div (2 \times 220) = .045$. In the output, notice the global statistic and its p-value change because of the fewer haplotypes, while the haplotype-specific scores do not change.

```
> 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)

--------------------------------------------------------------------------------
                 Haplotype Effect Model: additive
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
                      Global Score Statistics
--------------------------------------------------------------------------------

global-stat = 20.66451, df = 7, p-val = 0.0043

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

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

6.7 Score Statistic Dependencies: the *eps.svd* parameter (*UPDATED*)

A new parameter added for *haplo.score* is *eps.svd*. It is used in calculating the generalized inverse of the variance matrix of the score vector, performed by the *Ginv* function. The 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 thus 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 score statistics, and likewise, there is disparity in calculating the rank of the variance matrix.

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 insntatnces, 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 fixed when we increased epsilon for *Ginv*. We have added the *eps.svd* parameter to give the user control of epsilon for the singular value cutoff, and we have set the default at $1e-5$, versus the previous default of $1e-6$.

6.8 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.

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

```
 -------------------------------------------------------------
                 Haplotype Effect Model: dominant
 -------------------------------------------------------------
 -------------------------------------------------------------
                      Global Score Statistics
 -------------------------------------------------------------

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

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

      DQB DRB B  Hap-Freq Hap-Score p-val
 [1,] 21  3   8  0.10408  -2.23872  0.02517
 [2,] 31  4   44 0.02849  -2.13233  0.03298
 [3,] 51  1   44 0.01731  -0.99357  0.32043
 [4,] 63  13  44 0.01606  -0.84453  0.39837
 [5,] 63  2   7  0.01333  -0.50736  0.6119
 [6,] 32  4   60 0.0306   -0.46606  0.64118
 [7,] 21  7   44 0.02332  -0.41942  0.67491
 [8,] 62  2   44 0.01367  -0.26221  0.79316
 [9,] 62  2   18 0.01545  -0.21493  0.82982
[10,] 51  1   27 0.01505  0.01539   0.98772
```

6.9 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[1]. 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*= $\frac{1}{4}$ 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*.

```
> 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
 --------------------------------------------------------------
 --------------------------------------------------------------
                  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
 --------------------------------------------------------------

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

```
[16,] 21  7   44 0.02332  0.84562   0.39776 0.39796
[17,] 31  4   44 0.02849  2.50767   0.01215 0.01161
[18,] 21  3   8  0.10408  3.77763   0.00016 0.00035
```

7 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. 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 haplotype posterior probabilities. See Lake et al.[7] for details.

7.1 Preparing the *data.frame* for *haplo.glm* (*UPDATED*)

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 (see *lm* or *glm*). So, a *data.frame* must be defined, and this object must contain the trait, a special kind of genotype matrix (*geno.glm* for this example) that contains the genotypes of the marker loci, and optionally other covariates and weights. The key features of this *data.frame* are in how we set up a genotype matrix specific for use in *haplo.glm*. We prepare *geno.glm* below 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 initial 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 *geno.glm*, look at the attribute of it, then create a *data.frame* object (*my.data*) for use in *haplo.glm*. We now require the use of *setupGeno*, but we no longer need the *allele.lev* and *miss.val* parameters that were required in previous versions of *haplo.glm*.

```
> 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"
[12] "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"
[12] "42" "44" "45" "46" "47" "48" "49" "50" "51" "52" "55"
[23] "56" "57" "58" "60" "61" "62" "63" "70"
```

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

7.2 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 happlotype. 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 7.6.3.

7.3 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.

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

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

Coefficients:
                  coef    se t.stat    pval
(Intercept)     1.0644 0.343  3.105 0.00217
male            0.0974 0.155  0.627 0.53119
geno.glm.17     0.2802 0.435  0.643 0.52062
geno.glm.34    -0.3171 0.343 -0.923 0.35684
geno.glm.77     0.2217 0.361  0.614 0.54014
geno.glm.78     1.1414 0.384  2.974 0.00328
geno.glm.100    0.5556 0.364  1.525 0.12871
geno.glm.138    0.9823 0.303  3.239 0.00139
geno.glm.rare   0.3976 0.182  2.186 0.02992

Haplotypes:
              DQB DRB  B hap.freq
geno.glm.17     21   7 44   0.0229
geno.glm.34     31   4 44   0.0286
geno.glm.77     32   4 60   0.0302
geno.glm.78     32   4 62   0.0239
geno.glm.100    51   1 35   0.0301
geno.glm.138    62   2  7   0.0502
geno.glm.rare    *   *  *   0.7100
haplo.base      21   3  8   0.1041
```

Explanation of Results

The above table for *Coefficients* lists the estimated regression coefficient (*coef*), its standard error (*se*), the corresponding t-statistic (*t.stat*), and p-value (*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 = 0.01136* 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 7.6.2.

7.4 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.

```
> fit.inter <- haplo.glm(formula = y ~ male * geno.glm,
+     family = gaussian, data = my.data, na.action = "na.geno.keep",
+     locus.label = label, control = haplo.glm.control(haplo.min.count = 10))
> print(fit.inter)
```

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

Coefficients:
                      coef    se  t.stat    pval
(Intercept)         0.9754 0.523  1.8661 0.06347
male                0.2581 0.674  0.3832 0.70201
geno.glm.17         0.1444 0.545  0.2648 0.79144
geno.glm.34        -0.1716 0.668 -0.2570 0.79744
geno.glm.77         0.8052 0.650  1.2398 0.21649
geno.glm.78         0.4956 0.566  0.8760 0.38208
geno.glm.100        0.5231 0.481  1.0883 0.27776
geno.glm.138        1.1570 0.423  2.7337 0.00681
geno.glm.rare       0.4555 0.287  1.5859 0.11432
male:geno.glm.17    0.5087 0.875  0.5812 0.56176
male:geno.glm.34   -0.2814 0.786 -0.3581 0.72063
```

```
male:geno.glm.77   -0.9008 0.791 -1.1386 0.25618
male:geno.glm.78    1.2638 0.771  1.6385 0.10287
male:geno.glm.100   0.0507 0.775  0.0655 0.94785
male:geno.glm.138  -0.4459 0.619 -0.7203 0.47218
male:geno.glm.rare -0.0979 0.372 -0.2631 0.79272

Haplotypes:
              DQB DRB  B hap.freq
geno.glm.17    21   7 44   0.0235
geno.glm.34    31   4 44   0.0285
geno.glm.77    32   4 60   0.0306
geno.glm.78    32   4 62   0.0241
geno.glm.100   51   1 35   0.0301
geno.glm.138   62   2  7   0.0505
geno.glm.rare   *   *  *   0.7086
haplo.base     21   3  8   0.1041
```

Explanation of Results

The listed results are as explained under section 7.3. 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.

7.5 Regression for a Binomial Trait

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

```
> fit.bin <- haplo.glm(y.bin ~ male + geno.glm,
+     family = binomial, data = my.data, na.action = "na.geno.keep",
+     locus.label = label, control = haplo.glm.control(haplo.min.count = 10))
> print(fit.bin)
```

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

Coefficients:
                coef    se t.stat     pval
(Intercept)    1.546 0.655  2.361 0.019137
male          -0.480 0.331 -1.452 0.148055
```

```
geno.glm.17   -0.723 0.801 -0.902 0.367986
geno.glm.34    0.364 0.680  0.536 0.592782
geno.glm.77   -0.988 0.733 -1.349 0.178829
geno.glm.78   -1.409 0.854 -1.650 0.100519
geno.glm.100  -2.591 1.128 -2.297 0.022594
geno.glm.138  -2.716 0.852 -3.186 0.001661
geno.glm.rare -1.261 0.354 -3.565 0.000451

Haplotypes:
              DQB DRB  B hap.freq
geno.glm.17    21   7 44   0.0230
geno.glm.34    31   4 44   0.0284
geno.glm.77    32   4 60   0.0306
geno.glm.78    32   4 62   0.0235
geno.glm.100   51   1 35   0.0298
geno.glm.138   62   2  7   0.0518
geno.glm.rare   *   *  *   0.7088
haplo.base     21   3  8   0.1041
```

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.[12] 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. [12] and Epstein and Satten [5] for further details.

7.5.1 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 or S-PLUS 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 occuring, increase the minimum count or minimum frequency.

7.6 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.

7.6.1 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 1: Coding haplotype covariates in a model matrix

Hap - Pair	additive	dominant	recessive
Heterozygous	1	1	0
Homozygous	2	1	1

Note that in a recessive model, the haplotype effects are estimated only from subjects who are homozygous 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 6.8, 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.

```
> fit.dom <- haplo.glm(y ~ male + geno.glm, family = gaussian,
+     data = my.data, na.action = "na.geno.keep",
+     locus.label = label, control = haplo.glm.control(haplo.effect = "dominant
+         haplo.min.count = 8))
> print(fit.dom)
```

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

Coefficients:
                 coef    se t.stat     pval
(Intercept)    1.6493 0.373  4.416 1.61e-05
male           0.0797 0.157  0.507 6.13e-01
geno.glm.17   -0.0604 0.423 -0.143 8.87e-01
geno.glm.34   -0.6650 0.364 -1.827 6.91e-02
geno.glm.77   -0.0734 0.347 -0.212 8.33e-01
geno.glm.78    0.8537 0.364  2.344 2.00e-02
geno.glm.100   0.2470 0.346  0.715 4.76e-01
geno.glm.138   0.6729 0.282  2.389 1.78e-02
geno.glm.rare  0.1120 0.340  0.329 7.42e-01

Haplotypes:
              DQB DRB  B hap.freq
geno.glm.17    21   7 44   0.0230
geno.glm.34    31   4 44   0.0286
geno.glm.77    32   4 60   0.0302
geno.glm.78    32   4 62   0.0239
geno.glm.100   51   1 35   0.0300
geno.glm.138   62   2  7   0.0502
geno.glm.rare   *   *  *   0.7100
haplo.base     21   3  8   0.1041
```

7.6.2 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.

```
> 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
 [6] 0.05097906
```

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

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

 Coefficients:
                 coef    se t.stat   pval
 (Intercept)   -0.431 1.359 -0.317 0.7513
 male          -0.480 0.331 -1.452 0.1481
 geno.glm.4     0.988 0.733  1.349 0.1788
 geno.glm.17    0.266 1.025  0.259 0.7958
 geno.glm.34    1.353 0.922  1.466 0.1440
 geno.glm.78   -0.421 1.043 -0.404 0.6870
 geno.glm.100  -1.602 1.301 -1.232 0.2194
 geno.glm.138  -1.727 1.032 -1.674 0.0957
 geno.glm.rare -0.273 0.683 -0.399 0.6904

 Haplotypes:
               DQB DRB  B hap.freq
 geno.glm.4     21   3  8   0.1041
 geno.glm.17    21   7 44   0.0230
 geno.glm.34    31   4 44   0.0284
 geno.glm.78    32   4 62   0.0235
 geno.glm.100   51   1 35   0.0298
 geno.glm.138   62   2  7   0.0518
 geno.glm.rare   *   *  *   0.7088
 haplo.base     32   4 60   0.0306
```

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 3.4. In this document, we use the same seed before making *fit.bin* and *fit.bin.base77*.

7.6.3 More On Rare Haplotype (*UPDATED*)

Another notable control parameter is the minimum frequency 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.

```
> fit.bin.rare02 <- haplo.glm(y.bin ~ geno.glm,
+     family = binomial, data = my.data, na.action = "na.geno.keep",
+     locus.label = label, control = haplo.glm.control(haplo.freq.min = 0.02,
+         haplo.min.info = 0.02))
> print(fit.bin.rare02)
```

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

Coefficients:
                coef   se t.stat   pval
(Intercept)    1.241 1.32  0.937 0.3496
geno.glm.17   -0.607 1.56 -0.388 0.6983
geno.glm.34    0.319 1.27  0.252 0.8016
geno.glm.77   -1.072 1.37 -0.784 0.4337
geno.glm.78   -1.359 4.37 -0.311 0.7558
geno.glm.100  -2.398 2.09 -1.149 0.2520
geno.glm.138  -2.610 1.50 -1.735 0.0842
geno.glm.rare -1.223  NaN    NaN    NaN

Haplotypes:
              DQB DRB  B hap.freq
```

```
geno.glm.17      21    7 44    0.0230
geno.glm.34      31    4 44    0.0284
geno.glm.77      32    4 60    0.0306
geno.glm.78      32    4 62    0.0235
geno.glm.100     51    1 35    0.0298
geno.glm.138     62    2  7    0.0519
geno.glm.rare     *    *  *    0.7087
haplo.base       21    3  8    0.1040
```

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.

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: *haplo.score.merge*

When analyzing a qualitative trait, such as binary, 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, nlines = 10)
```

```
------------------------------------------------------------
        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  -2.19387 0.02824  0.05098 0.06789 0.01587
2   51   1 35  -1.58421 0.11315  0.03018 0.03754 0.00907
3   63  13  7  -1.56008 0.11874  0.01655 0.02176      NA
4   21   7  7  -1.47495 0.14023  0.01246 0.01969      NA
5   32   4  7  -1.00091 0.31687  0.01678 0.02628 0.00794
```

```
6    32    4 62  -0.67990 0.49657  0.02349 0.01911      NA
7    51    1 27  -0.66509 0.50599  0.01505 0.01855 0.00907
8    31   11 35  -0.58380 0.55936  0.01754 0.01982 0.01587
9    31   11 51  -0.43721 0.66196  0.01137 0.01321      NA
10   51    1 44   0.00826 0.99341  0.01731 0.01595 0.00000
```

Explanation of Results

The first column is a row index, the next columns (3 in this example) illustrate the haplotype, the *Hap.Score* column is the score statistic and *p.val* the corresponding χ^2 p-value. *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 (*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=TRUE*, which prints 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 frequencyies.

8.2 Case-Control Haplotype Analysis: *haplo.cc* (*UPDATED*)

It is possible to combine the results of *haplo.score*, *haplo.group*, and *haplo.glm* for case-control data, all performed within *haplo.cc*. The function peforms a score test and a glm on the same haplotypes. The parameters that determine which haplotypes are used are *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.

```
> 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 = 0.02))
> 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)
-----------------------------------------------------------
    DQB DRB  B Hap-Score   p-val pool.hf control.hf case.hf
147  62   2  7    -2.103 0.03546  0.0490     0.0679   0.016
98   51   1 35    -1.583 0.11344  0.0302     0.0376   0.009
78   32   4  7    -1.393 0.16349  0.0227     0.0263   0.008
77   32   4 62    -0.496 0.62001  0.0212     0.0191      NA
76   32   4 60     0.028 0.97762  0.0307     0.0315   0.024
16   21   7 44     1.069 0.28516  0.0217     0.0175   0.048
52   31   4 44     2.516 0.01186  0.0285     0.0150   0.063
11   21   3  8     3.776 0.00016  0.1042     0.0693   0.190
1    21   1  8        NA      NA  0.0023     0.0033      NA
2    21  10  8        NA      NA  0.0023     0.0032      NA
3    21   2 18        NA      NA  0.0023     0.0032      NA
4    21   2  7        NA      NA  0.0023     0.0032      NA
5    21   3 18        NA      NA  0.0046     0.0067      NA
6    21   3 35        NA      NA  0.0057     0.0065      NA
7    21   3 44        NA      NA  0.0036     0.0033   0.016
8    21   3 49        NA      NA  0.0023         NA      NA
9    21   3 57        NA      NA  0.0024         NA      NA
10   21   3 70        NA      NA  0.0023         NA      NA
12   21   4 62        NA      NA  0.0045     0.0064      NA
13   21   7 13        NA      NA  0.0108         NA   0.024
14   21   7 18        NA      NA  0.0025         NA      NA
15   21   7 35        NA      NA  0.0024         NA   0.008
17   21   7 45        NA      NA  0.0023     0.0032      NA
18   21   7 50        NA      NA  0.0045     0.0032   0.008
19   21   7 57        NA      NA  0.0023     0.0064      NA
    glm.eff OR.lower    OR OR.upper
147     Eff   0.0138 0.072     0.38
98      Eff   0.0097 0.086     0.76
78      Eff   0.0047 0.058     0.72
77      Eff   0.0517 0.281     1.53
76      Eff   0.0763 0.318     1.32
16      Eff   0.1253 0.661     3.48
52      Eff   0.3425 1.318     5.07
11     Base       NA 1.000       NA
```

```
1          R   0.1443 0.290     0.58
2          R   0.1443 0.290     0.58
3          R   0.1443 0.290     0.58
4          R   0.1443 0.290     0.58
5          R   0.1443 0.290     0.58
6          R   0.1443 0.290     0.58
7          R   0.1443 0.290     0.58
8          R   0.1443 0.290     0.58
9          R   0.1443 0.290     0.58
10         R   0.1443 0.290     0.58
12         R   0.1443 0.290     0.58
13         R   0.1443 0.290     0.58
14         R   0.1443 0.290     0.58
15         R   0.1443 0.290     0.58
17         R   0.1443 0.290     0.58
18         R   0.1443 0.290     0.58
19         R   0.1443 0.290     0.58
```

```
> names(cc.hla)
```

```
[1] "cc.df"          "group.count"  "score.lst"
[4] "fit.lst"        "ci.prob"      "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*.

```
> 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)

  start.loc score.global.p global.p.sim max.p.sim
1         1        0.21550           NA        NA
2         2        0.09366           NA        NA
3         3        0.39042           NA        NA
4         4        0.48771           NA        NA
5         5        0.13747           NA        NA
6         6        0.15064           NA        NA
7         7        0.11001           NA        NA
8         8        0.00996           NA        NA
9         9        0.04255           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 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 8^{th}, 9^{th}, and 10^{th} x-axis tick marks.

```
> plot.haplo.score.slide(score.slide.gaus)
```

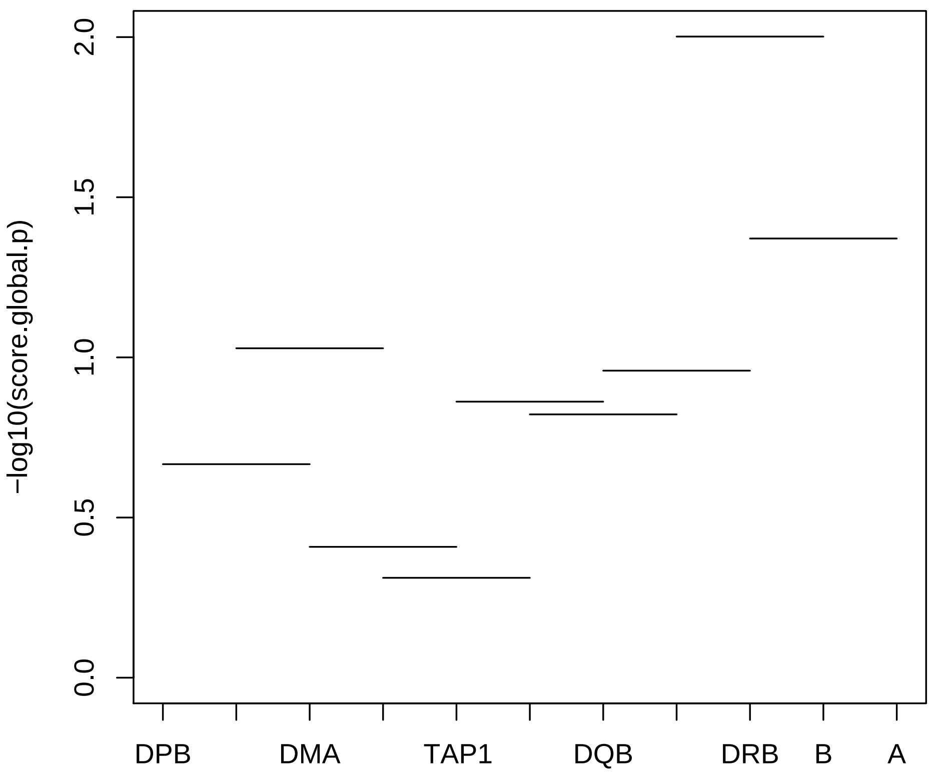

Figure 2: Global p-values for sub-haplotypes; Gaussian Response

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 [3]. 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 of 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.

```
> 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))
> many.haps <- (1:length(y.bin))[hla.summary[, 4] >
+     50000]
> geno.scan <- geno.11[-many.haps, ]
> y.scan <- y.bin[-many.haps]
> 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:
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
```

```
============================================================
          loc-1 loc-2 loc-3 loc-4 loc-5 loc-6 loc-7 loc-8
sim.p-val     0     0     0     0     0     0     0     0
          loc-9 loc-10 loc-11
sim.p-val     0      0      0


      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 6.9. 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 (UPDATED)*

Another approach for choosing loci for haplotype associations is by *seqhap*, as described in Yu and Schaid, 2007 [14]. 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 [9]. For each marker locus, three tests are provided:

- **single locus**, the traditional single-locus χ^2_1 test of association,
- **sequential haplotype**, based on a haplotype test for sequentially chosen loci,
- **sequential summary**, based on the sum of a series of conditional χ^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 6.9, 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. In S-PLUS, example data is available in *seqhap.dat* and chromosome positions *seqhap.pos*. In R, these objects must be loaded using *data()*. The following example runs *seqhap* with default settings for permutations and threshold parameters.

```
> data(seqhap.dat)
> data(seqhap.pos)
> y <- seqhap.dat$disease
> geno <- seqhap.dat[, -1]
> pos <- seqhap.pos$pos
> seqhap.out <- seqhap(y = y, geno = geno, pos = pos,
+      miss.val = c(0, NA), r2.threshold = 0.95,
+      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
        chi.stat perm.point.p asym.point.p
 loc-1   1.22062   0.27729741      0.26924
 loc-2   1.35462   0.23245526      0.24447
 loc-3   5.20288   0.02010859      0.02255
 loc-4   3.36348   0.05972250      0.06666
 loc-5   3.55263   0.06153227      0.05945
 loc-6   0.39263   0.53026342      0.53092
 loc-7   5.54913   0.01829881      0.01849
 loc-8   3.74740   0.05469535      0.05289
 loc-9   0.03602   0.85682687      0.84947
 loc-10  1.99552   0.17313493      0.15777


 ============================================================
                       Sequential Scan
 ============================================================
 Loci Combined in Sequential Analysis
```

```
seq-loc-1 1
seq-loc-2 2 3 4 5
seq-loc-3 3 4 5
seq-loc-4 4 3
seq-loc-5 5
seq-loc-6 6 7
seq-loc-7 7
seq-loc-8 8
seq-loc-9 9
seq-loc-10 10


============================================================
                 Sequential Haplotype Test
============================================================
Regional permuted P-value based on sequential haplotype test is  0.016489
           hap.stat df perm.point.p asym.point.p
seq-loc-1   1.22062  1  0.310878745      0.26924
seq-loc-2  24.16488 12  0.027950935      0.01932
seq-loc-3  19.78808  6  0.005228232      0.00302
seq-loc-4  14.95765  3  0.003016288      0.00185
seq-loc-5   3.55263  1  0.096722300      0.05945
seq-loc-6   5.45723  2  0.114216771      0.06531
seq-loc-7   5.54913  1  0.038608486      0.01849
seq-loc-8   3.74740  1  0.103961392      0.05289
seq-loc-9   0.03602  1  0.867886588      0.84947
seq-loc-10  1.99552  1  0.219384677      0.15777


============================================================
                  Sequential Summary Test
============================================================
Regional permuted P-value based on sequential summary test is  0.0032174
           sum.stat df perm.point.p asym.point.p
seq-loc-1   1.22062  1 0.3108787452      0.26924
seq-loc-2  21.15360  4 0.0008043435      0.00030
seq-loc-3  18.65769  3 0.0008043435      0.00032
seq-loc-4  14.61897  2 0.0020108586      0.00067
seq-loc-5   3.55263  1 0.1033581339      0.05945
seq-loc-6   5.43826  2 0.1150211140      0.06593
seq-loc-7   5.54913  1 0.0386084858      0.01849
```

```
seq-loc-8   3.74740  1 0.1041624774      0.05289
seq-loc-9   0.03602  1 0.8678865876      0.84947
seq-loc-10  1.99552  1 0.2193846773      0.15777
```

Explanation of Results

The output from this example first shows *n.sim*, the number of permutations needed for precision on the regional p-values. Next, in the printed results, the first section (*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 χ^2_1 distribution. The second section (*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 (*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 (*Sequential Summary Test*) shows similar information for the sequential summary 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 summary asymptotic p-value), and *"sum.sim"* (sequential summary simulated p-value). The other parameter option is *single*, indicating whether to plot a line for the single-locus tests.

```
> plot(seqhap.out, pval = "hap", single = TRUE,
+      las = 2)
```

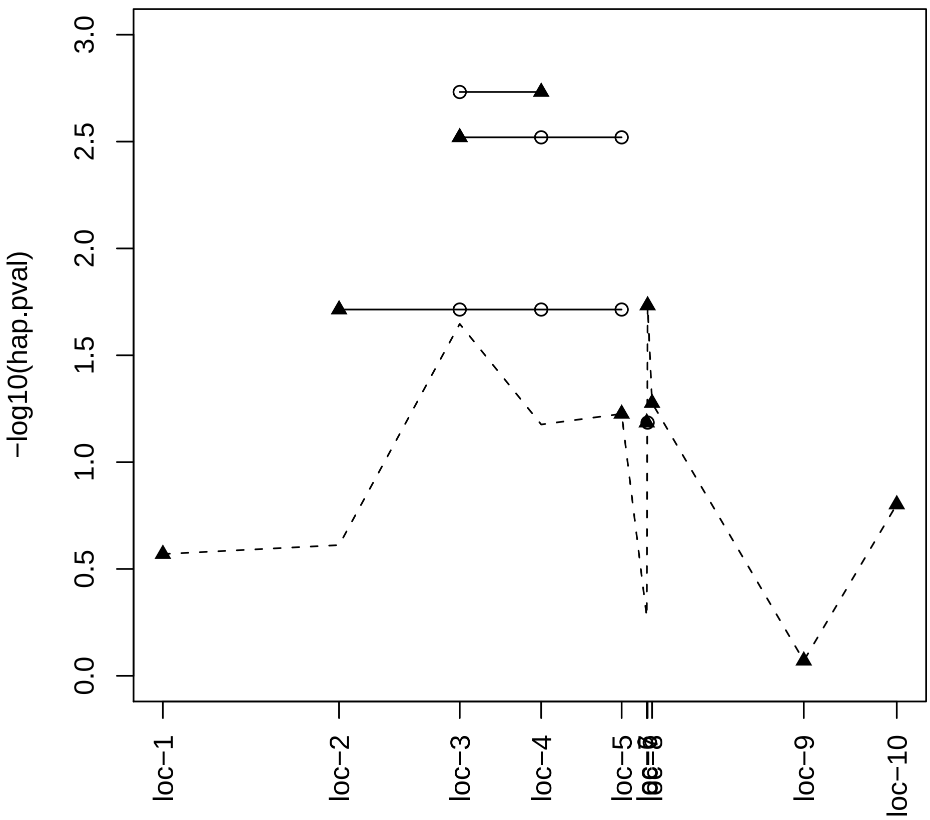

Figure 3: Plot p-values for sequential haplotype scan and single-locus tests

8.6 Creating Haplotype Effect Columns: *haplo.design*

In some instances, the desired model for haplotype effects is not possible with the methods given in *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 *haplo.design*, which will set up an expected haplotype design matrix from a *haplo.em* object, to create columns that can be used to model haplotype effects in other modeling functions.

The function *haplo.design* first creates a design marix 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 [15], and using this haplotype design matrix in a regression model is asymptotically equivalent to the score statistics from *haplo.score* (Xie and Stram 2005 [13]). 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 [8] 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 *min.count*. The columns are named by the code they are assigned in *haplo.em*.

```
> hap.effect.frame <- haplo.design(save.em)
> names(hap.effect.frame)

 [1] "hap.4"   "hap.13"  "hap.17"  "hap.34"  "hap.50"
 [6] "hap.55"  "hap.69"  "hap.77"  "hap.78"  "hap.99"
[11] "hap.100" "hap.102" "hap.138" "hap.140" "hap.143"
[16] "hap.155" "hap.162" "hap.165"

> hap.effect.frame[1:10, 1:8]

   hap.4    hap.13    hap.17 hap.34 hap.50 hap.55 hap.69
1      0 0.0000000 0.0000000      0      0      0      0
2      0 0.1253234 0.8746766      0      0      0      0
3      0 0.0000000 0.0000000      0      0      0      0
4      0 0.2862131 0.7137869      0      0      0      0
5      0 0.0000000 0.0000000      0      0      1      0
6      1 0.0000000 1.0000000      0      0      0      0
7      0 0.0000000 0.0000000      0      0      0      0
8      0 0.0000000 0.0000000      0      0      0      0
9      0 0.0000000 0.0000000      0      0      0      0
10     0 0.0000000 0.0000000      0      0      0      0
```

```
    hap.77
1        0
2        0
3        0
4        0
5        0
6        0
7        0
8        0
9        0
10       0
```

Additionally, *haplo.design* gives the user flexibility to make a more specific design matrix with the following parameters:

- **hapcodes:** codes assigned in the *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"*.

```
> 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 0.0000000
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
```

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

 Deviance Residuals:
     Min       1Q   Median       3Q      Max
 -2.3261  -1.0749  -0.0656   1.0448   2.3904

 Coefficients:
              Estimate Std. Error t value Pr(>|t|)
 (Intercept)  1.913834   0.229577   8.336  9.1e-15 ***
 male         0.048588   0.155290   0.313   0.7547
 age         -0.002651   0.011695  -0.227   0.8209
 hap.4       -0.405530   0.195857  -2.071   0.0396 *
 hap.138      0.584480   0.261763   2.233   0.0266 *
 ---
 Signif. codes:  0 ,***, 0.001 ,**, 0.01 ,*, 0.05 ,., 0.1 , , 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

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 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 $\frac{a_i(a_i-1)}{2}$ 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($g()$)

Number of missing alleles	Homozygous function $f(a_i)$	Heterozygous function $g(a_i)$
1	1	$a_i - 1$
2	a_i	$\frac{a_i(a_i-1)}{2}$

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

$$a_i 2^x + \left[\frac{a_i(a_i-1)}{2}\right] 2^{x+1} \tag{1}$$

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

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

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

$$TotalPairs = \sum_{j=0}^{m} \sum_{combo} C(combo, j) \tag{3}$$

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^{nd} and 3^{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)

m	$j = 0\, of\, m$	$j = 1\, of\, m$	$j = 2\, of\, m$	$j = 3\, of\, m$
0	2^x			
1	$f(a_1)2^x$	$g(a_1)2^{x+1}$		
2	$f(a_1)f(a_2)2^x$	$g(a_1)f(a_2)2^{x+1}$ $f(a_1)g(a_2)2^{x+1}$	$g(a_1)g(a_2)2^{x+1}$	
3	$f(a_1)f(a_2)f(a_3)2^x$	$g(a_1)f(a_2)f(a_3)2^{x+1}$ $f(a_1)g(a_2)f(a_3)2^{x+1}$ $f(a_1)f(a_2)g(a_3)2^{x+1}$	$g(a_1)g(a_2)f(a_3)2^{x+2}$ $g(a_1)f(a_2)g(a_3)2^{x+2}$ $f(a_1)g(a_2)g(a_3)2^{x+2}$	$g(a_1)g(a_2)g(a_3)2^{x+2}$

References

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

[2] Cheng R, Ma JZ, Wright FA, Lin S, Gau X, Wang D, Elston RC, Li MD. (2003) Nonparametric disequilibrium mapping of functional sites using haplotypes of multiple tightly linked single-nucleotide polymorphism markers. *Genetics* 164:1175-1187.

[3] Cheng R, Ma JZ, Elston RC, Li MD. (2005) Fine Mapping Functional Sites or Regions from Case-Control Data Using Haplotypes of Multiple Linked SNPs. *Annals of Human Genetics* 69: 102-112.

[4] Clayton, David. Personal web page, software list. April 1, 2004. <http://www-gene.cimr.cam.ac.uk/clayton/software/>.

[5] Epstein MP, Satten GA (2003) Inference on haplotype effects in case-control studies using unphased genotype data. *Am J Hum Genet* 73: 1316-1329.

[6] Harrell, FE. Regression Modeling Strategies, Springer-Verlag, NY, 2001.

[7] 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.

[8] Lin DY, Zeng D (2006) Likelihood-based inference on haplotype effects in genetic association studies. *J Am Stat Assoc* Vol 101, No. 473.

[9] Mantel N, Haenszel W. (1959) Statistical aspects of the analysis of data from retrospective studies of disease. *J Nat Cancer Inst* 22:719-48.

[10] 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.

[11] Schaid DJ (2005) Power and Sample Size for Testing Associations of Haplotypes with Complex Traits. *Annals of Human Genetics* 70:116-130.

[12] 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.

[13] Xie R, Stram DO (2005) Asymptotic equivalence between two score tests for haplotype-specific risk in general linear models. *Gen Epi* 29:166-170.

[14] Yu Z, Schaid DJ. (2007) Sequential haplotype scan methods for association analysis. To appear in *Gen Epi*.

[15] Zaykin DV, Westfall PH, Young SS, Karnoub MA, Wagner MJ, Ehm MG (2002) Testing Association of Statistically Inferred Haplotypes with Discreet and Continuous Traits in Samples of Unrelated Individuals. *Human Heredity* 53:79-91.