Package 'haplo.stats'

October 9, 2026

Version 1.9.9.1

Date 2026-09-10

Title Statistical Analysis of Haplotypes with Traits and Covariates when Linkage Phase is Ambiguous

Description Routines for the analysis of indirectly measured haplotypes. The statistical methods assume that all subjects are unrelated and that haplotypes are ambiguous (due to unknown linkage phase of the genetic markers). The main functions are: haplo.em(), haplo.glm(), haplo.score(), and haplo.power(); all of which have detailed examples in the vignette.

License GPL (>= 2)

Depends R (>= 4.4.0), methods, stats, graphics, arsenal, MASS

Suggests knitr, testthat

VignetteBuilder knitr

URL https://CRAN.R-project.org/package=haplo.stats

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Schaid Daniel [aut], Jason P. Sinnwell [aut, cre]

Maintainer Jason P. Sinnwell <sinnwell.jason@mayo.edu>

Repository CRAN

Date/Publication 2026-09-10 20:00:02 UTC

Contents

anova.haplo.glm 3
chisq.power 4
dglm.fit 5
f.power 5
find.haplo.beta.qt 6
fitted.haplo.glm 7
geno.count.pairs 7

geno1to2 . . . 8
get.hapPair . . . 9
Ginv . . . 11
haplo.cc . . . 12
haplo.design . . . 14
haplo.em . . . 15
haplo.em.control . . . 18
haplo.em.fitter . . . 19
haplo.glm . . . 20
haplo.glm.control . . . 25
haplo.group . . . 27
haplo.hash . . . 28
haplo.model.frame . . . 29
haplo.power.cc . . . 30
haplo.power.qt . . . 32
haplo.scan . . . 35
haplo.score . . . 37
haplo.score.merge . . . 41
haplo.score.slide . . . 42
hapPower.demo . . . 45
hla.demo . . . 46
locator.haplo . . . 47
locus . . . 48
louis.info . . . 49
na.geno.keep . . . 50
plot.haplo.score . . . 50
plot.haplo.score.slide . . . 51
plot.seqhap . . . 53
print.haplo.cc . . . 54
print.haplo.em . . . 55
print.haplo.group . . . 56
print.haplo.scan . . . 57
print.haplo.score . . . 57
print.haplo.score.merge . . . 58
print.haplo.score.slide . . . 59
printBanner . . . 59
residuals.haplo.glm . . . 60
score.sim.control . . . 61
seqhap . . . 62
seqhap.dat . . . 65
setupGeno . . . 66
summary.haplo.em . . . 67
summary.haplo.glm . . . 68
summaryGeno . . . 69
vcov.haplo.glm . . . 70
x.sexcheck . . . 70

Index **72**

anova.haplo.glm *Analysis of variance for haplo.glm model fit*

Description

Perform an analysis of variance between two haplo.glm model fits using the deviances from the fitted objects

Usage

```
## S3 method for class 'haplo.glm'
anova(object, ..., dispersion=NULL, test="Chisq")
```

Arguments

object	A haplo.glm or glm object
...	More model fits to compare against the fit in the first argument
dispersion	the dispersion parameter for the fitting family. By default it is obtained from the object(s)
test	character string for the test of model comparison. Only "Chisq" supported for haplo.glm objects

Details

Uses print.anova for the displayed result

Value

A data.frame of the anova class, with these columns: Df, Deviance, Resid.Df, Resid.Dev, p-value

See Also

haplo.glm

Examples

```
data(hla.demo)
geno <- as.matrix(hla.demo[,c(17,18,21:24)])
keep <- !apply(is.na(geno) | geno==0, 1, any) # SKIP THESE THREE LINES
hla.demo <- hla.demo[keep,]                   # IN AN ANALYSIS
geno <- geno[keep,]                           #
attach(hla.demo)
label <-c("DQB","DRB","B")
y <- hla.demo$resp
y.bin <- 1*(hla.demo$resp.cat=="low")

# set up a genotype array as a model.matrix for inserting into data frame
# Note that hla.demo is a data.frame, and we need to subset to columns
```

```
# of interest. Also also need to convert to a matrix object, so that
# setupGeno can code alleles and convert geno to 'model.matrix' class.

 geno <- setupGeno(geno, miss.val=c(0,NA))

  # geno now has an attribute 'unique.alleles' which must be passed to
  # haplo.glm as allele.lev=attributes(geno)$unique.alleles, see below

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


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

 anova.haplo.glm(glmfit.gaus, fit.gaus)
```

---

chisq.power *Power and sample size for the chi-square distribution*

Description

Power and sample size for the chi-square distribution given non-centrality, degrees of freedom, alpha, N (for chisq.power), and power (for chisq.sample.size)

Usage

```
chisq.power(n, nc, df, alpha)
chisq.power.dif(n, nc, df, alpha, power)
chisq.sample.size(nc, df=df, alpha, power, lower=20, upper=100000)
```

Arguments

n	sample size (for power)
nc	non-centrality parameter
df	degrees of freedom
alpha	type-I error rate
power	desired power (for sample size)
lower	lower bound for search space for sample size
upper	upper bound for search space for sample size

Value

power, the difference in power from target power, and sample size, respectively for the three different functions

`dglm.fit` *Internal functions for the HaploStats package. See the help file for the main functions (haplo.em, haplo.score, haplo.glm) for details on some of these functions.*

Description

Internal function for the HaploStats package

`f.power` *Power and sample size for the F distribution*

Description

Power and sample size for the F distribution given non-centrality, degrees of freedom, alpha, N (for f.power), and power (for f.sample.size)

Usage

```
f.power(n, nc, df1, alpha)
f.power.dif(n, nc, df1, alpha, power)
f.sample.size(nc, df1, alpha, power, lower=20, upper=10000)
```

Arguments

`n`	sample size
`nc`	non-centrality parameter
`df1`	degrees of freedom for numerator of f distribution
`alpha`	type-I error
`power`	desired power (for sample size)
`lower`	lower limit for search space for sample size solution
`upper`	upper limit for search space for sample size solution

Value

power, the difference in power from target power, and sample size, respectively for the three functions, assuming an F distribution for the test statistic

`find.haplo.beta.qt` *Find beta coefficients for risk haplotypes, for specified r2*

Description

Find betas for risk haplotypes and intercept (beta for base.index haplotype) with a given r2

Usage

```
find.haplo.beta.qt(haplo, haplo.freq, base.index, haplo.risk, r2, y.mu=0, y.var=1)
find.beta.qt.phase.known(beta.size, haplo.risk, base.index, haplo,
haplo.freq, r2, y.mu, y.var)
find.intercept.qt.phase.known(beta.no.intercept, base.index, haplo, haplo.freq, y.mu)
```

Arguments

`haplo`	matrix of haplotypes, with rows the different haplotypes and columns the alleles of the haplotypes. For H haplotypes of L loci, haplo has dimension H x L.
`haplo.freq`	vector of length H for the population haplotype frequencies (corresponding to the rows of haplo)
`base.index`	integer index of the haplotype considered to be the base-line for logistic regression (index between 1 and H); often, the most common haplotype is chosen for the base-line.
`haplo.risk`	vector of relative risks for haplotypes
`r2`	correlation coefficient
`y.mu`	mean of y, a quantitative trait
`y.var`	variance of y, a quantitative trait
`beta.size`	beta values for risk haplotypes in find.beta.qt.phase.known
`beta.no.intercept`	beta vector for haplotypes for quantitative trait, excluding the beta for intercept

Value

beta estimates for haplotypes or intercept

fitted.haplo.glm *Fitted values from haplo.glm fit*

Description

The fitted values for each person, collapsed over their expanded fitted values due to their multiple possible haplotype pairs

Usage

```
## S3 method for class 'haplo.glm'
fitted(object, ...)
```

Arguments

object	A haplo.glm object
...	Optional arguments for the method

Details

Many of the subjects in a haplo.glm fit are expanded in the model matrix with weights used to reflect the posterior probability of the subject's haplotype pairs given their genotype. The working fitted values within the fitted object are from this expanded model matrix, and the fitted values from this method are calculated from the weighted fitted value for the subject across all their haplotype pairs.

Value

vector of fitted values

See Also

haplo.glm

geno.count.pairs *Counts of Total Haplotype Pairs Produced by Genotypes*

Description

Provide a count of all possible haplotype pairs for each subject, according to the phenotypes in the rows of the geno matrix. The count for each row includes the count for complete phenotypes, as well as possible haplotype pairs for phenotypes where there are missing alleles at any of the loci.

Usage

```
geno.count.pairs(geno)
```

Arguments

geno	Matrix of alleles, 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 geno has 2*K columns. Rows represent all observed alleles for each subject, their phenotype.

Details

When a subject has no missing alleles, and has h heterozygous sites, there are 2**(h-1) haplotype pairs that are possible ('**'=power). For loci with missing alleles, we consider all possible pairs of alleles at those loci. Suppose that there are M loci with missing alleles, and let the vector V have values 1 or 0 acccording to whether these loci are imputed to be heterozygous or homozygous, respectively. The length of V is M. The total number of possible states of V is 2**M. Suppose that the vector W, also of length M, provides a count of the number of possible heterozygous/homozygous states at the loci with missing data. For example, if one allele is missing, and there are K possible alleles at that locus, then there can be one homozygous and (K-1) heterozygous genotypes. If two alleles are missing, there can be K homozygous and K(K-1)/2 heterozygous genotypes. Suppose the function H(h+V) counts the total number of heterozygous sites among the loci without missing data (of which h are heterozygous) and the imputed loci (represented by the vector V). Then, the total number of possible pairs of haplotypes can be respresented as SUM(W*H(h+V)), where the sum is over all possible values for the vector V.

Value

Vector where each element gives a count of the number haplotype pairs that are consistent with a subject's phenotype, where a phenotype may include 0, 1, or 2 missing alleles at any locus.

See Also

haplo.em, summaryGeno

Examples

```
data(hla.demo)
genohla <- hla.demo[,c(17,18,21:24)]
geno <- setupGeno(genohla)
count.geno <- geno.count.pairs(geno)
print(count.geno)
```

---

geno1to2 *convert genotype matrix from 1-column 2-column*

Description

convert 1-column genotype matrix to 2-column genotype matrix, converting from a minor allele count (0,1,2) to (1/1, 1/2, 2/2) where 2 is the minor allele. (not supported for x-linked markers)

Usage

```
geno1to2(geno, locus.label=NULL)
```

Arguments

geno	1-column representation of genotype matrix for 2-allele loci. Values are 0, 1, or 2, usually the count of minor alleles
locus.label	Vector of labels for loci, If a locus name is "A", its columns will be "A.1" and "A.2"

Value

a 2-column genotype matrix

Examples

```
geno1 <- matrix(c(0,0,1,
                  1,0,2,
                  2,1,0), ncol=3, byrow=TRUE)
geno1to2(geno1, locus.label=c("A", "B", "C"))

## demonstrate how NA and 3 will be coded
geno1[1,3] <- NA
geno1[1,1] <- 3
geno1to2(geno1)
```

---

get.hapPair *Get a list of objects for haplotype pairs*

Description

Get a list of objects for modeling haplotype pairs from a set of unique haplotypes and their frequencies, given the baseline haplotype

Usage

```
get.hapPair(haplo, haplo.freq, base.index)
```

Arguments

haplo	matrix of haplotypes, with rows the different haplotypes and columns the alleles of the haplotypes. For H haplotypes of L loci, haplo has dimension H x L.
haplo.freq	vector of length H for the population haplotype frequencies (corresponding to the rows of haplo)
base.index	integer index of the haplotype considered to be the base-line for logistic regression (index between 1 and H); often, the most common haplotype is chosen for the base-line.

Value

list with components:

p.g	Genotype probability under Hardy-Weinberg Equilibrium, where the genotype is the haplotype pair
x.haplo	Design matrix for all pairs of haplotypes, excluding the baseline haplotype. Effects are coded to an additive effect for the haplotypes.
haplo.indx	two-column matrix containing the indices for the haplotypes in x.haplo. The indices are the row of the haplotype in haplo.

Examples

```
haplo <- rbind(
               c(     1,    2,    2,    1,    2),
               c(     1,    2,    2,    1,    1),
               c(     1,    1,    2,    1,    1),
               c(     1,    2,    1,    1,    2),
               c(     1,    2,    2,    2,    1),
               c(     1,    2,    1,    1,    1),
               c(     1,    1,    2,    2,    1),
               c(     1,    1,    1,    1,    2),
               c(     1,    2,    1,    2,    1),
               c(     1,    1,    1,    2,    1),
               c(     2,    2,    1,    1,    2),
               c(     1,    1,    2,    1,    2),
               c(     1,    1,    2,    2,    2),
               c(     1,    2,    2,    2,    2),
               c(     2,    2,    2,    1,    2),
               c(     1,    1,    1,    1,    1),
               c(     2,    1,    1,    1,    1),
               c(     2,    1,    2,    1,    1),
               c(     2,    2,    1,    1,    1),
               c(     2,    2,    1,    2,    1),
               c(     2,    2,    2,    1,    1))
dimnames(haplo)[[2]] <- paste("loc", 1:ncol(haplo), sep=".")
haplo <- data.frame(haplo)

haplo.freq <- c(0.170020121, 0.162977867, 0.123742455, 0.117706237, 0.097585513, 0.084507042,
            0.045271630, 0.039235412, 0.032193159, 0.019114688, 0.019114688, 0.013078471,
            0.013078471, 0.013078471, 0.013078471, 0.006036217, 0.006036217, 0.006036217,
                0.006036217, 0.006036217, 0.006036217)

hPair <- get.hapPair(haplo, haplo.freq, base.index=1)
names(hPair)
dim(hPair$x.haplo)
```

`Ginv` *Compute Generalized Inverse of Input Matrix*

Description

Singular value decomposition (svd) is used to compute a generalized inverse of input matrix.

Usage

```
Ginv(x, eps=1e-6)
```

Arguments

`x`	A matrix.
`eps`	minimum cutoff for singular values in svd of x

Details

The svd function uses the LAPACK standard library to compute the singular values of the input matrix, and the rank of the matrix is determined by the number of singular values that are at least as large as max(svd)*eps, where eps is a small value. For S-PLUS, the Matrix library is required (Ginv loads Matrix if not already done so).

Value

List with components:

`Ginv`	Generalized inverse of x.
`rank`	Rank of matrix x.

References

Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical recipes in C. The art of scientific computing. 2nd ed. Cambridge University Press, Cambridge.1992. page 61.

Anderson, E., et al. (1994). LAPACK User's Guide, 2nd edition, SIAM, Philadelphia.

See Also

svd

Examples

```
# for matrix x, extract the generalized inverse and
# rank of x as follows
    x <- matrix(c(1,2,1,2,3,2),ncol=3)
    save <- Ginv(x)
    ginv.x <- save$Ginv
    rank.x <- save$rank
```

haplo.cc *Haplotype Association Analysis in a Case-Control design*

Description

Combine results from haplo.score, haplo.group, and haplo.glm for case-control study designs. Analyze the association between the binary (case-control) trait and the haplotypes relevant to the unrelated individuals' genotypes.

Usage

```
haplo.cc(y, geno, x.adj=NA, locus.label=NA, ci.prob=0.95,
         miss.val=c(0,NA), weights=NULL, eps.svd=1e-5, simulate=FALSE,
         sim.control=score.sim.control(), control=haplo.glm.control())
```

Arguments

y	Vector of trait values, must be 1 for cases and 0 for controls.
geno	Matrix of alleles, 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 ncol(geno) = 2*K. Rows represent alleles for each subject.
x.adj	Matrix of non-genetic covariates used to adjust the score statistics. Note that intercept should not be included, as it will be added in this function.
ci.prob	Probability level for confidence interval on the Odds Ratios of each haplotype to span the true value.
locus.label	Vector of labels for loci, of length K (see definition of geno matrix)
miss.val	Vector of codes for missing values of alleles
weights	the weights for observations (rows of the data frame). By default, all observations are weighted equally. One use is to correct for over-sampling of cases in a case-control sample.
eps.svd	epsilon value for singular value cutoff; to be used in the generalized inverse calculation on the variance matrix of the score vector. The degrees of freedom for the global score test is 1 less than the number of haplotypes that are scored (k-1). The degrees of freedom is calculated from the rank of the variance matrix for the score vector. In some instances of numeric instability, the singular value decomposition indicates full rank (k). One remedy has been to give a larger epsilon value.
simulate	Logical: if [F]alse, no empirical p-values are computed; if [T]rue, simulations are performed within haplo.score. Specific simulation parameters can be controlled in the sim.control parameter list.
sim.control	A list of control parameters to determine how simulations are performed for simulated p-values. The list is created by the function score.sim.control and the default values of this function can be changed as desired. See score.sim.control for details.

`control` A list of control parameters for managing the execution of haplo.cc. The list is created by the function haplo.glm.control, which also manages control parameters for the execution of haplo.em.

Details

All function calls within haplo.cc are for the analysis of association between haplotypes and the case-control status (binomial trait). No additional covariates may be modeled with this function. Odd Ratios are in reference to the baseline haplotype. Odds Ratios will change if a different baseline is chosen using haplo.glm.control.

Value

A list including the haplo.score object (score.lst), vector of subject counts by case and control group (group.count), haplo.glm object (fit.lst), confidence interval probability (ci.prob), and a data frame (cc.df) with the following components:

`haplotypes`	The first K columns contain the haplotypes used in the analysis.
`Hap-Score`	Score statistic for association of haplotype with the binary trait.
`p-val`	P-value for the haplotype score statistic, based on a chi-square distribution with 1 degree of freedom.
`sim.p.val`	Vector of p-values for score.haplo, based on simulations in haplo.score (omitted when simulations not performed). P-value of score.global based on simulations (set equal to NA when simulate=F).
`pool.hf`	Estimated haplotype frequency for cases and controls pooled together.
`control.hf`	Estimated haplotype frequency for control group subjects.
`case.hf`	Estimated haplotype frequency for case group subjects.
`glm.eff`	The haplo.glm function modeled the haplotype effects as: baseline (Base), additive haplotype effect (Eff), or rare haplotypes pooled into a single group (R).
`OR.lower`	Lower limit of the Odds Ratio Confidence Interval.
`OR`	Odds Ratio based on haplo.glm model estimated coefficient for the haplotype.
`OR.upper`	Upper limit of the Odds Ratio Confidence Interval.

References

Schaid et al Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. Score tests for association of traits with haplotypes when linkage phase is ambiguous. Amer J Hum Genet. 70 (2002): 425-434.

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

See Also

`haplo.em`, `haplo.score`, `haplo.group`, `haplo.score.merge`, `haplo.glm` `print.haplo.cc`

Examples

```
#  For a genotype matrix geno.test, case/control vector y.test
#  The function call will be like this
# cc.test <- haplo.cc(y.test, geno.test, locus.label=locus.label, haplo.min.count=3, ci.prob=0.95)
#
```

`haplo.design` *Build a design matrix for haplotypes*

Description

Build a design matrix for haplotypes estimated from a haplo.em object.

Usage

```
haplo.design(obj, haplo.effect="additive", hapcodes=NA, min.count=5, haplo.base=NA)
```

Arguments

`obj`	an object created from haplo.em
`haplo.effect`	The "effect" pattern of haplotypes on the response. This parameter determines the coding for scoring the haplotypes. Valid coding options for heterozygous and homozygous carriers of a haplotype are "additive" (1, 2, respectively), "dominant" (1,1, respectively), and "recessive" (0, 1, respectively).
`hapcodes`	codes assigned in haplo.em, corresponding to the row numbers in the haplotypes matrix item in `obj`
`min.count`	The minimum number of estimated counts of the haplotype in the sample in order for a haplotype to be included in the design matrix.
`haplo.base`	code for which haplotype will be the reference group, or to be considered the baseline of a model. The code is the row number of the `haplotypes` matrix in `obj`. This haplotype is removed from the design matrix.

Details

First a matrix is made for the possible haplotypes for each person, coded for the haplo.effect, weighted by the posterior probability of those possible haplotypes per person, and then collapsed back to a single row per person.

Value

Matrix of columns for haplotype effects. Column names are "hap.k" where k is the row number of the unique haplotypes within the haplo.em object's "haplotypes" item.

See Also

`haplo.em`

Examples

```
###-----------------------------------------------
### See the user manual for more complete examples
###-----------------------------------------------

data(hla.demo)
attach(hla.demo)

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

keep <- !apply(is.na(geno) | geno==0, 1, any)

save.em.keep <- haplo.em(geno=geno[keep,], locus.label=label)

save.df <- haplo.design(save.em.keep, min.count=10)
dim(save.df)

names(save.df)
save.df[1:10,]
```

`haplo.em` *EM Computation of Haplotype Probabilities, with Progressive Insertion of Loci*

Description

For genetic marker phenotypes measured on unrelated subjects, with linkage phase unknown, compute maximum likelihood estimates of haplotype probabilities. Because linkage phase is unknown, there may be more than one pair of haplotypes that are consistent with the oberved marker phenotypes, so posterior probabilities of pairs of haplotypes for each subject are also computed. Unlike the usual EM which attempts to enumerate all possible pairs of haplotypes before iterating over the EM steps, this "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 algorithm reduces to the usual EM algorithm.

Usage

```
haplo.em(geno, locus.label=NA, miss.val=c(0, NA), weight, control=
           haplo.em.control())
```

Arguments

geno	matrix of alleles, 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 ncol(geno) = 2*K. Rows represent the alleles for each subject.
locus.label	vector of labels for loci.
miss.val	vector of values that represent missing alleles in geno.
weight	weights for observations (rows of geno matrix).
control	list of control parameters. The default is constructed by the function haplo.em.control. The default behavior of this function results in the following parameter settings: loci.insert.order=1:n.loci, insert.batch.size=min(4,n.loci), min.posterior= 0.0001, tol=0.00001, max.iter=500, random.start=0 (no random start), iseed=NULL (no saved seed to start random start), verbose=0 (no printout during EM iterations). See haplo.em.control for more details.

Details

The basis of this progressive insertion algorithm is from the sofware snphap by David Clayton. Although some of the features and control parameters of this S-PLUS version 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 alogrithm is implemented.

Value

list with components:

converge	indicator of convergence of the EM algorithm (1 = converge, 0 = failed).
lnlike	value of lnlike at last EM iteration (maximum lnlike if converged).
lnlike.noLD	value of lnlike under the null of linkage equilibrium at all loci.
lr	likelihood ratio statistic to test the final lnlike against the lnlike that assumes complete linkage equilibrium among all loci (i.e., haplotype frequencies are products of allele frequencies).
df.lr	degrees of freedom for likelihood ratio statistic. The df for the unconstrained final model is the number of non-zero haplotype frequencies minus 1, and the df for the null model of complete linkage equilibrium is the sum, over all loci, of (number of alleles - 1). The df for the lr statistic is df[unconstrained] - df[null]. This can result in negative df, if many haplotypes are estimated to have zero frequency, or if a large amount of trimming occurs, when using large values of min.posterior in the list of control parameters.
hap.prob	vector of mle's of haplotype probabilities. The ith element of hap.prob corresponds to the ith row of haplotype.
locus.label	vector of labels for loci, of length K (see definition of input values).
subj.id	vector of id's for subjects used in the analysis, based on row number of input geno matrix. If subjects are removed, then their id will be missing from subj.id.

rows.rem	now defunct, but set equal to a vector of length 0, to be compatible with other functions that check for rows.rem.
indx.subj	vector for row index of subjects after expanding to all possible pairs of haplotypes for each person. If indx.subj=i, then i is the ith row of geno. If the ith subject has n possible pairs of haplotypes that correspond to their marker genotype, then i is repeated n times.
nreps	vector for the count of haplotype pairs that map to each subject's marker genotypes.
max.pairs	vector of maximum number of pairs of haplotypes per subject that are consistent with their marker data in the matrix geno. The length of max.pairs = nrow(geno). This vector is computed by geno.count.pairs.
hap1code	vector of codes for each subject's first haplotype. The values in hap1code are the row numbers of the unique haplotypes in the returned matrix haplotype.
hap2code	similar to hap1code, but for each subject's second haplotype.
post	vector of posterior probabilities of pairs of haplotypes for a person, given their marker phenotypes.
haplotype	matrix of unique haplotypes. Each row represents a unique haplotype, and the number of columns is the number of loci.
control	list of control parameters for algorithm. See haplo.em.control

Note

Sorted order of haplotypes with character alleles is system-dependent, and can be controlled via the LC_COLLATE locale environment variable. Different locale settings can cause results to be non-reproducible even when controlling the random seed.

See Also

setupGeno, haplo.em.control

Examples

```
data(hla.demo)
attach(hla.demo)
geno <- hla.demo[,c(17,18,21:24)]
label <-c("DQB","DRB","B")
keep <- !apply(is.na(geno) | geno==0, 1, any)

save.em.keep <- haplo.em(geno=geno[keep,], locus.label=label)

# warning: output will not exactly match

print.haplo.em(save.em.keep)
```

`haplo.em.control` *Create the Control Parameters for the EM Computation of Haplotype Probabilities, with Progressive Insertion of Loci*

Description

Create a list of parameters that control the EM algorithm for estimating haplotype frequencies, based on progressive insertion of loci. Non-default parameters for the EM algorithm can be set as parameters passed to haplo.em.control.

Usage

```
haplo.em.control(loci.insert.order=NULL, insert.batch.size = 6,
                              min.posterior = 1e-09, tol = 1e-05,
                              max.iter=5000, random.start=0, n.try = 10,
                              iseed=NULL, max.haps.limit=2e6, verbose=0)
```

Arguments

`loci.insert.order`
: Numeric vector with specific order to insert the loci. If this value is NULL, the insert order will be in sequential order (1, 2, ..., No. Loci).

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

`min.posterior`
: Minimum posterior probability of a 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. If all markers in low LD, we recommend using the default. If markers have at least moderate LD, can increase this value to use less memory.

`tol`
: If the change in log-likelihood value between EM steps is less than the tolerance (tol), it has converged.

`max.iter`
: Maximum number of iterations allowed for the EM algorithm before it stops and prints an error. If the error is printed, double max.iter.

`random.start`
: If random.start = 0, then the inititial starting values of the posteriors for the first EM attempt will be based on assuming equal posterior probabilities (conditional on genotypes). If random.start = 1, then the initial starting values of the first EM attempt will be based on assuming a uniform distribution for the initial posterior probabilities.

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

`iseed`
: An integer or a saved copy of .Random.seed. This allows simulations to be reproduced by using the same initial seed.

`max.haps.limit` Maximum number of haplotypes for the input genotypes. It is used as the amount of memory to allocate in C for the progressive-insertion E-M steps. 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 no longer request more.

`verbose` Logical, if TRUE, print procedural messages to the screen. If FALSE, do not print any messages.

Details

The default is to use n.try = 10. If this takes too much time, it may be worthwhile to decrease n.try. Other tips for computing haplotype frequencies for a large number of loci, particularly if some have many alleles, is to decrease the batch size (insert.batch.size), increase the memory (max.haps.limit), and increase the probability of trimming off rare haplotypes at each insertion step (min.posterior).

Value

A list of the parameters passed to the function.

See Also

`haplo.em`, `haplo.score`

Examples

```
# This is how it is used within haplo.score
#    > score.gauss <- haplo.score(resp, geno, trait.type="gaussian",
#    >           em.control=haplo.em.control(insert.batch.size = 2, n.try=1))
```

---

haplo.em.fitter *Compute engine for haplotype EM algorithm*

Description

For internal use within the haplo.stats library

Usage

```
haplo.em.fitter(n.loci, n.subject, weight, geno.vec, n.alleles,
                max.haps, max.iter, loci.insert.order, min.posterior,
                tol, insert.batch.size, random.start, iseed1, iseed2,
                iseed3, verbose)
```

Arguments

n.loci	number of loci in genotype matrix
n.subject	number of subjects in the sample
weight	numeric weights
geno.vec	vectorized genotype matrix
n.alleles	numeric vector giving number of alleles at each marker
max.haps	maximum unique haplotypes in the sample
max.iter	maximum iterations to perform in the fitter
loci.insert.order	order to insert loci for progressive insertion
min.posterior	after insertion and maximization, discard haplotype pairs per person that do not meet minimum posterior prob
tol	convergence tolerance for E-M steps
insert.batch.size	number of markers to insert per batch
random.start	logical; if TRUE, allow for random starting values of haplotype frequencies
iseed1	random seed for algorithm
iseed2	random seed for algorithm
iseed3	random seed for algorithm
verbose	logical, print long, verbose output from E-M steps?

Details

For internal use within the haplo.stats library

---

haplo.glm *GLM Regression of Trait on Ambiguous Haplotypes*

Description

Perform glm regression of a trait on haplotype effects, allowing for ambiguous haplotypes. This method performs an iterative two-step EM, with the posterior probabilities of pairs of haplotypes per subject used as weights to update the regression coefficients, and the regression coefficients used to update the posterior probabilities.

Usage

```
haplo.glm(formula=formula(data), family=gaussian, data=parent.frame(),
          weights, na.action="na.geno.keep", start=NULL,
          locus.label=NA, control=haplo.glm.control(),
          method="glm.fit", model=TRUE, x=FALSE, y=TRUE,
          contrasts=NULL, ...)
```

Arguments

formula	a formula expression as for other regression models, of the form response ~ predictors. For details, see the documentation for lm and formula.
family	a family object. This is a list of expressions for defining the link, variance function, initialization values, and iterative weights for the generalized linear model. Supported families are: gaussian, binomial, poisson. Currently, only the logit link is implemented for binimial.
data	a data frame in which to interpret the variables occurring in the formula. A CRITICAL element of the data frame is the matrix of genotypes, denoted here as "geno", although an informative name should be used in practice. This geno matrix is actually a matrix of alleles, 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 ncol(geno) = 2*K. Rows represent the alleles for each subject. It is also CRITICAL that this matrix is defined as a model.matrix, so the columns of the matrix are packaged together into a single matrix object. If geno is a matrix of alleles, then before adding it to the data frame, use the setupGeno() function, which will assign this correct class. The function will also recode alleles to numeric starting from 1, while saving the original alleles in the unique.alleles attribute. This attribute is required in haplo.glm.
weights	the weights for observations (rows of the data frame). By default, all observations are weighted equally.
na.action	a function to filter missing data. This is applied to the model.frame. The default value of na.action=na.geno.keep will keep observations with some (but not all) missing alleles, but exclude observations missing any other data (e.g., response variable, other covariates, weight). The EM algorithm for ambiguous haplotypes accounts for missing alleles. Similar to the usual glm, na.fail creates an error if any missing values are found, and a third possible alternative is na.exclude, which deletes observations that contain one or more missing values for any data, including alleles.
start	a vector of initial values on the scale of the linear predictor.
locus.label	vector of labels for loci.
control	list of control parameters. The default is constructed by the function haplo.glm.control. The items in this list control the regression modeling of the haplotypes (e.g., additive, dominant, recessive effects of haplotypes; which haplotype is chosen as the baseline for regression; how to handle rare haplotypes; control of the glm function - maximum number of iterations), and the EM algorithm for estimating initial haplotype frequencies. See haplo.glm.control for details.
method	currently, glm.fit is the only method allowed.
model	logical, if model=TRUE, the model.frame is returned.
x	logical, if x=TRUE, the model.matrix is returned.
y	logical, if y=TRUE, the response variable is returned.
contrasts	currently ignored
...	other arguments that may be passed - currently ignored.

Details

To properly prepare the data frame, the genotype matrix must be processed by setupGeno, and then included in the data frame with the response and other variables.

For binomial family, the initialization of values gives warnings if non-integer number of successes, which is a concern in these models because of the weights of posterior probability of each haplotype pair per subject. We supress the warnings by defining a haplo.binomial family, which we use if family=binomial is used.

Value

An object of class "haplo.glm" is returned. The output object from haplo.glm has all the components of a glm object, with a few more. It is important to note that some of the returned components correpond to the "expanded" version of the data. This means that each observation is expanded into the number of terms in the observation's posterior distribution of haplotype pairs, given the marker data. For example, when fitting the response y on haplotype effects, the value of y[i], for the ith observation, is replicated m[i] times, where m[i] is the number of pairs of haplotypes consistent with the observed marker data. The returned components that are expanded are indicated below by [expanded] in the definition of the component.

These expanded components may need to be collapsed, depending on the objective of the user. For example, when considering the influence of an observation, it may make sense to examine the expanded residuals for a single observation, perhaps plotted against the haplotypes for that observation. In contrast, it would not be sensible to plot all residuals against non-genetic covariates, without first collapsing the expanded residuals for each observation. To collapse, one can use the average residual per observation, weighted according to the posterior probabilities. The appropriate weight can be computed as wt = weight.expanded * haplo.post.info[[post]]. Then, the weighted average can be calculated as with(fit, tapply(residuals * wt, haplo.post.info[["indx"]], sum).

`coefficients`	the coefficients of the linear.predictors, which multiply the columns of the model matrix. The names of the coefficients are the names of the column of the model matrix. For haplotype coefficients, the names are the concatentation of name of the geno matrix with a haplotype number. The haplotype number corresponds to the index of the haplotype. The default print will show the coefficients with haplotype number, along with the alleles that define the haplotype, and the estimated haplotype frequency. If the model is over-determined there will be missing values in the coefficients corresponding to inestimable coefficients.
`residuals`	[expanded] residuals from the final weighted least squares fit; also known as working residuals, these are typically not interpretable without rescaling by the weights (see glm.object and residuals.haplo.glm).
`fitted.values`	[expanded] fitted mean values, obtained by transforming linear.predictors using the inverse link function (see glm.object).
`effects`	[expaded] orthogonal, single-degree-of-freedom effects (see lm.object).
`R`	the triangular factor of the decomposition (see lm.object).
`rank`	the computed rank (number of linearly independent columns in the model matrix), which is the model degrees of freedom - see lm.object.
`assign`	the list of assignments of coefficients (and effects) to the terms in the model (see lm.object).

`df.residual` [expanded] number of degrees of freedom for residuals, corresponding to the expanded data.

`prior.weights` [expanded] input weights after expanding according to the number of pairs of haplotypes consistent with an observation's marker genotype data.

`family` a 3 element character vector giving the name of the family, the link and the variance function; mainly for printing purposes.

`linear.predictors` [expanded] linear fit, given by the product of the model matrix and the coefficients. In a glm, eta.

`deviance` up to a constant, minus twice the maximized log-likelihood. Similar to the residual sum of squares.

`null.deviance` the deviance corresponding to the model with no predictors.

`call` an image of the call that produced the object, but with the arguments all named and with the actual formula included as the formula argument.

`iter` the number of IRLS iterations used to compute the estimates, for the last step of the EM fit of coefficients.

`y` expanded response.

`contrasts` a list containing sufficient information to construct the contrasts used to fit any factors occurring in the model (see lm.object).

`lnlike` log-likelihood of the fitted model.

`lnlike.null` log-likelihood of the null model (intercept-only).

`lrt` likelihood ratio test statistic to test whether all coefficients (excepet intercept) are zero: 2*(lnlike - lnlike.null)

`terms` an object of mode expression and class term summarizing the formula, but not complete for the final model. Because this does not represent expansion of the design matrix for the haplotypes, it is typically not of direct relevance to users.

`control` list of all control parameters

`haplo.unique` the data.frame of unique haplotypes

`haplo.base` the index of the haplotype used as the base-line for the regression model. To see the actual haplotype definition, use the following: with(fit, haplo.unique[haplo.base,]), where fit is the saved haplo.glm object (e.g., fit <- haplo.glm(y ~ geno, ...)).

`haplo.freq` the final estimates of haplotype frequencies, after completing EM steps of updating haplotype frequencies and regression coefficients. The length of haplo.freq is the number of rows of haplo.unique, and the order of haplo.freq is the same as that for the rows of haplo.unique. So, the frequencies of the unique haplotypes can be viewed as with(fit, cbind(haplo.unique, haplo.freq)).

`haplo.freq.init` the initial estimates of haplotype frequencies, based on the EM algorithm for estimating haplotype frequencies, ingnoring the trait. These can be compared with haplo.freq, to see the impact of using the regression model to update the haplotype frequencies.

`converge.em` T/F whether the EM-glm steps converged

haplo.common	the indices of the haplotypes determined to be "common" enough to estimate their corresponding regression coefficients.
haplo.rare	the indices of all the haplotypes determined to be too rare to estimate their specific regression coefficients.
haplo.rare.term	T/F whether the "rare" term is included in the haplotype regression model.
haplo.names	the names of the coefficients that represent haplotype effects.
haplo.post.info	a data.frame of information regarding the posterior probabilites. The columns of this data.frame are: indx (the index of the input obsevation; if the ith observation is repeated m times, then indx will show m replicates of i; hence, indx will correspond to the "expanded" observations); hap1 and hap2 (the indices of the haplotypes; if hap1=j and hap2=k, then the two haplotypes in terms of alleles are haplo.unique[j,] and haplo.unique[k,] from the fitted object); post.init (the initial posterior probability, based on haplo.freq.init); post (the final posterior probability, based on haplo.freq).
x	the model matrix, with [expanded] rows, if x=T.
info	the observed information matrix, based on Louis' formula. The upper left submatrix is for the regression coefficient, the lower right submatrix for the haplotype frequencies, and the remaining is the information between regression coefficients and haplotype frequencies.
var.mat	the variance-covariance matrix of regression coefficients and haplotype frequencies, based on the inverse of info. Upper left submatrix is for regression coefficients, lower right submatrix for haplotype frequencies.
haplo.elim	the indices of the haplotypes eliminated from the info and var.mat matrices because their frequencies are less than haplo.min.info (the minimum haplotype frequency required for computation of the information matrix - see haplo.glm.control)
missing	a matrix of logical values, indicating whether rows of data were removed for missing values in either genotype matrix (genomiss) or any other variables (yxmiss), such as y, other covariates, or weights.
rank.info	rank of information (info) matrix.

References

Lake S, Lyon H, Silverman E, Weiss S, Laird N, Schaid D (2002) Estimation and tests of haplotype-environment interaction when linkage phase is ambiguous. Human Heredity 55:56-65.

See Also

haplo.glm.control, haplo.em, haplo.model.frame

Examples

```
cat(" FOR REGULAR USAGE, DO NOT DISCARD GENOTYPES WITH MISSING VALUES\n")
cat(" WE ONLY SUBSET BY keep HERE SO THE EXAMPLES RUN FASTER\n")

 data(hla.demo)
```

```
 geno <- as.matrix(hla.demo[,c(17,18,21:24)])
 keep <- !apply(is.na(geno) | geno==0, 1, any) # SKIP THESE THREE LINES
 hla.demo <- hla.demo[keep,]                  # IN AN ANALYSIS
 geno <- geno[keep,]                          #
 attach(hla.demo)
 label <-c("DQB","DRB","B")
 y <- hla.demo$resp
 y.bin <- 1*(hla.demo$resp.cat=="low")

# set up a genotype array as a model.matrix for inserting into data frame
# Note that hla.demo is a data.frame, and we need to subset to columns
# of interest. Also also need to convert to a matrix object, so that
# setupGeno can code alleles and convert geno to 'model.matrix' class.

 geno <- setupGeno(geno, miss.val=c(0,NA))

  # geno now has an attribute 'unique.alleles' which must be passed to
  # haplo.glm as allele.lev=attributes(geno)$unique.alleles, see below

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

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

`haplo.glm.control` *Create list of control parameters for haplo.glm*

Description

Create a list of control pararameters for haplo.glm. If no parameters are passed to this function, then all default values are used.

Usage

```
haplo.glm.control(haplo.effect="add", haplo.base=NULL,
                  haplo.min.count=NA, haplo.freq.min=.01,
                  sum.rare.min=0.001, haplo.min.info=0.001,
                  keep.rare.haplo=TRUE,
                  eps.svd=sqrt(.Machine$double.eps),
                  glm.c=glm.control(maxit=500),
                  em.c=haplo.em.control())
```

Arguments

haplo.effect	the "effect" of a haplotypes, which determines the covariate (x) coding of haplotypes. Valid options are "additive" (causing x = 0, 1, or 2, the count of a particular haplotype), "dominant" (causing x = 1 if heterozygous or homozygous carrier of a particular haplotype; x = 0 otherwise), and "recessive" (causing x = 1 if homozygous for a particular haplotype; x = 0 otherwise).
haplo.base	the index for the haplotype to be used as the base-line for regression. By default, haplo.base=NULL, so that the most frequent haplotype is chosen as the base-line.
haplo.min.count	The minimum number of expected counts for a haplotype from the sample to be included in the model. The count is based on estimated haplotype frequencies. Suggested minimum is 5.
haplo.freq.min	the minimum haplotype frequency for a haplotype to be included in the regression model as its own effect. The haplotype frequency is based on the EM algorithm that estimates haplotype frequencies independent of trait.
sum.rare.min	the sum of the "rare" haplotype frequencies must be larger than sum.rare.min in order for the pool of rare haplotypes to be included in the regression model as a separate term. If this condition is not met, then the rare haplotypes are pooled with the base-line haplotype (see keep.rare.haplo below).
haplo.min.info	the minimum haplotype frequency for determining the contribution of a haplotype to the observed information matrix. Haplotypes with less frequency are dropped from the observed information matrix. The haplotype frequency is that from the final EM that iteratively updates haplotype frequencies and regression coefficients.
keep.rare.haplo	TRUE/FALSE to determine if the pool of rare haplotype should be kept as a separate term in the regression model (when keep.rare.haplo=TRUE), or pooled with the base-line haplotype (when keep.rare.haplo=FALSE).
eps.svd	argument to be passed to Ginv for the generalized inverse of the information matrix, helps to determine the number of singular values
glm.c	list of control parameters for the usual glm.control (see glm.control).
em.c	list of control parameters for the EM algorithm to estimate haplotype frequencies, independent of trait (see haplo.em.control).

Value

the list of above components

See Also

haplo.glm, haplo.em.control, glm.control

Examples

```
# NOT RUN
```

```
# using the data set up in the example for haplo.glm,
# the control function is used in haplo.glm as follows
#  > fit <- haplo.glm(y ~ male + geno, family = gaussian,
#  >          na.action="na.geno.keep",
#  >          data=my.data, locus.label=locus.label,
#  >          control = haplo.glm.control(haplo.min.count=5,
#  >          em.c=haplo.em.control(n.try=1)))
```

---

haplo.group *Frequencies for Haplotypes by Grouping Variable*

Description

Calculate maximum likelihood estimates of haplotype probabilities for the entire dataset and separately for each subset defined by the levels of a group variable. Only autosomal loci are considered.

Usage

```
haplo.group(group, geno, locus.label=NA,
            miss.val=0, weight=NULL,
            control=haplo.em.control())
```

Arguments

group	Group can be of logical, numeric, character, or factor class type.
geno	Matrix of alleles, 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 geno has 2*K columns. Rows represent all observed alleles for each subject.
locus.label	Vector of labels for loci, of length K (see definition of geno matrix).
miss.val	Vector of codes for allele missing values.
weight	weights for observations (rows of geno matrix). One reason to use is to adjust for disproportionate sample of sub-groups. Weights only used in the frequency calculation for the pooled subject.
control	list of control parameters for haplo.em (see haplo.em.control).

Details

Haplo.em is used to compute the maximum likelihood estimates of the haplotype frequencies for the total sample, then for each of the groups separately.

Value

list	A list as an object of the haplo.group class. The three elements of the list are described below.
group.df	A data frame with the columns described as follows. -haplotype: Names for the K columns for the K alleles in the haplotypes. -total: Estimated frequencies for haplotypes from the total sample. -group.name.i: Estimated haplotype frequencies for the haplotype if it occurs in the group referenced by 'i'. Frequency is NA if it doesn't occur for the group. The column name is the actual variable name joined with the ith level of that variable.
group.count	Vector containing the number of subjects for each level of the grouping variable.
n.loci	Number of loci occuring in the geno matrix.

References

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. "Score tests for association of traits with haplotypes when linkage phase is ambiguous." Amer J Hum Genet. 70 (2002): 425-434.

See Also

print.haplo.group, haplo.em

Examples

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

# remove any subjects with missing alleles for faster examples,
# but you may keep them in practice
  keep <- !apply(is.na(geno) | geno==0, 1, any)
  hla.demo <- hla.demo[keep,]
  geno <- geno[keep,]
  attach(hla.demo)

  y.ord <- as.numeric(resp.cat)
  y.bin <-ifelse(y.ord==1,1,0)
  group.bin <- haplo.group(y.bin, geno, miss.val=0)
  print.haplo.group(group.bin)
```

---

haplo.hash *Integer Rank Codes for Haplotypes*

Description

Create a vector of integer codes for the input matrix of haplotypes. The haplotypes in the input matrix are converted to character strings, and if there are C unique strings, the integer codes for the haplotypes will be 1, 2, ..., C.

Usage

```
haplo.hash(hap)
```

Arguments

hap	A matrix of haplotypes. If there are N haplotypes for K loci, hap have dimensions N x K.

Details

The alleles that make up each row in hap are pasted together as character strings, and the unique strings are sorted so that the rank order of the sorted strings is used as the integer code for the unique haplotypes.

Value

List with elements:

hash	Vector of integer codes for the input data (hap). The value of hash is the row number of the unique haplotypes given in the returned matrix hap.mtx.
hap.mtx	Matrix of unique haplotypes.

See Also

haplo.em

---

haplo.model.frame *Sets up a model frame for haplo.glm*

Description

For internal use within the haplo.stats library

Usage

```
haplo.model.frame(m, locus.label=NA, control=haplo.glm.control())
```

Arguments

m	model.frame from evaluated formula
locus.label	labels for loci in genotype matrix
control	control parameters for haplo.glm

Details

See haplo.glm description in help file and user manual

Value

A model frame with haplotypes modeled as effects

`haplo.power.cc` *Compute either power or sample size for haplotype associations in a case-control study.*

Description

For a given set of haplotypes, their population frequencies, and assumed logistic regression coefficients (log-odds-ratios per haplotype, assuming a log-additive model of haplotype effects), determine either the sample size (total number of subjects) to achieve a stated power or the power for a stated sample size.

Usage

```
haplo.power.cc(haplo, haplo.freq, base.index, haplo.beta, case.frac,
prevalence, alpha, sample.size=NULL, power=NULL)
```

Arguments

`haplo`	matrix of haplotypes, with rows the different haplotypes and columns the alleles of the haplotypes. For H haplotypes of L loci, haplo has dimension H x L.
`haplo.freq`	vector of length H for the population haplotype frequencies (corresponding to the rows of haplo)
`base.index`	integer index of the haplotype considered to be the base-line for logistic regression (index between 1 and H); often, the most common haplotype is chosen for the base-line.
`haplo.beta`	vector of length H for the haplotype effects: each beta is the log-odds-ratio for the corresponding haplotype effect. The base-line hapoltype should have a beta=0, as this base-line beta coefficient will be automatically calculated according to the haplotype frequencies, the other haplo.beta's, and the disease prevalence.
`case.frac`	fraction of cases in the total sample size (e.g., case.frac = .5 for typical case-control studies with equal numbers of cases and controls)
`prevalence`	popultaion disease prevalence (used to calculate the base-line intercept beta)
`alpha`	type-I error rate
`sample.size`	total sample size (if power is to be calcualted). Either sample.size or power must be specified, but not both.
`power`	desired power (if sample.size is to be calculated). Either sample.size or power must be specified, but not both.

Details

Asympotic power calcuations are based on the non-centrality parameter of a non-central chi-square distribution. This non-centrality parameter is determined by the specified regression coefficients (values in haplo.beta), as well as the distribution of haplotypes (determined by haplo.freq). To account for haplotypes with unknown phase, all possible haplotype pairs are enumerated, and the EM algorithm is used to determine the posterior probabilities of pairs of haplotypes, conditional on unphased genotype data. Because this function uses the function haplo.em, the number of possible haplotypes can be large when there is a large number of loci (i.e., large number of columns in the haplo matrix). If too large, the function haplo.em will run out of memory, making this function (haplo.power.cc) fail. If this occurs, then consider reducing the "size" of the haplotypes, by reducing the number of columns of haplo, and adjusting the corresponding vectors (e.g., haplo.freq, haplo.beta).

Value

list with components:

`ss.phased.haplo`
: sample size for phased haplotypes

`ss.unphased.haplo`
: sample size for unphased haplotypes

`power.phased.haplo`
: power for phased haplotypes

`power.unphased.haplo`
: power for unphased haplotypes

References

Schaid, DJ. Power and sample size for testing associations of haplotypes with complex traits. Ann Hum Genet (2005) 70:116-130.

See Also

`haplo.em` `haplo.power.qt`

Examples

```
haplo <- rbind(
               c(    1,    2,    2,    1,    2),
               c(    1,    2,    2,    1,    1),
               c(    1,    1,    2,    1,    1),
               c(    1,    2,    1,    1,    2),
               c(    1,    2,    2,    2,    1),
               c(    1,    2,    1,    1,    1),
               c(    1,    1,    2,    2,    1),
               c(    1,    1,    1,    1,    2),
               c(    1,    2,    1,    2,    1),
               c(    1,    1,    1,    2,    1),
               c(    2,    2,    1,    1,    2),
               c(    1,    1,    2,    1,    2),
```

```
                c(     1,    1,    2,    2,    2),
                c(     1,    2,    2,    2,    2),
                c(     2,    2,    2,    1,    2),
                c(     1,    1,    1,    1,    1),
                c(     2,    1,    1,    1,    1),
                c(     2,    1,    2,    1,    1),
                c(     2,    2,    1,    1,    1),
                c(     2,    2,    1,    2,    1),
                c(     2,    2,    2,    1,    1))
dimnames(haplo)[[2]] <- paste("loc", 1:ncol(haplo), sep=".")
haplo <- data.frame(haplo)

haplo.freq <- c(0.170020121, 0.162977867, 0.123742455, 0.117706237, 0.097585513, 0.084507042,
            0.045271630, 0.039235412, 0.032193159, 0.019114688, 0.019114688, 0.013078471,
            0.013078471, 0.013078471, 0.013078471, 0.006036217, 0.006036217, 0.006036217,
                0.006036217, 0.006036217, 0.006036217)

# define index for risk haplotypes (having alleles 1-1 at loci 2 and 3)
haplo.risk <- (1:nrow(haplo))[haplo$loc.2==1 & haplo$loc.3==1]

# define index for baseline haplotype
base.index <-  1

# specify OR for risk haplotypes
or <- 1.25

# determine beta regression coefficients for risk haplotypes

haplo.beta <- numeric(length(haplo.freq))
haplo.beta[haplo.risk] <-  log(or)

# Note that non-risk haplotypes have beta=0, as does the intercept
# (haplotype with base.index value).

# Compute total sample size for given power

haplo.power.cc(haplo, haplo.freq, base.index, haplo.beta, case.frac=.5,
prevalence=.1, alpha=.05, power=.8)

# Compute power for given sample size

haplo.power.cc(haplo, haplo.freq, base.index, haplo.beta, case.frac=.5,
prevalence=.1, alpha=.05, sample.size=11978)
```

`haplo.power.qt` *Compute either power or sample size for haplotype associations with a quantitative trait.*

Description

For a given set of haplotypes, their population frequencies, and assumed linear regression coefficients (additive model of haplotype effects on a quantitative trait), determine either the sample size to achieve a stated power or the power for a stated sample size.

Usage

```
haplo.power.qt(haplo, haplo.freq, base.index, haplo.beta, y.mu, y.var,
alpha, sample.size = NULL, power = NULL)
```

Arguments

haplo	matrix of haplotypes, with rows the different haplotypes and columns the alleles of the haplotypes. For H haplotypes of L loci, haplo has dimension H x L.
haplo.freq	vector of length H for the population haplotype frequencies (corresponding to the rows of haplo)
base.index	integer index of the haplotype considered to be the base-line for logistic regression (index between 1 and H); often, the most common haplotype is chosen for the base-line.
haplo.beta	vector of length H for the haplotype effects: each beta is the amount of expected change per haplotype from the base-line average, and the beta for the base-line (indexed by base.index) is the beta for the intercept.
y.mu	population mean of quantitative trait, y.
y.var	popultaion variance of quantitative trait, y.
alpha	type-I error rate
sample.size	sample size (if power is to be calcualted). Either sample.size or power must be specified, but not both.
power	desired power (if sample.size is to be calculated). Either sample.size or power must be specified, but not both.

Details

Asympotic power calcuations are based on the non-centrality parameter of a non-central F distribution. This non-centrality parameter is determined by the specified regression coefficients (values in haplo.beta), as well as the distribution of haplotypes (determined by haplo.freq). To account for haplotypes with unknown phase, all possible haplotype pairs are enumerated, and the EM algorithm is used to determine the posterior probabilities of pairs of haplotypes, conditional on unphased genotype data. Because this function uses the function haplo.em, the number of possible haplotypes can be large when there is a large number of loci (i.e., large number of columns in the haplo matrix). If too large, the function haplo.em will run out of memory, making this function (haplo.power.qt) fail. If this occurs, then consider reducing the "size" of the haplotypes, by reducing the number of columns of haplo, and adjusting the corresponding vectors (e.g., haplo.freq, haplo.beta).

Value

list with components:

`ss.phased.haplo`
: sample size for phased haplotypes

`ss.unphased.haplo`
: sample size for unphased haplotypes

`power.phased.haplo`
: power for phased haplotypes

`power.unphased.haplo`
: power for unphased haplotypes

References

Schaid, DJ. Power and sample size for testing associations of haplotypes with complex traits. Ann Hum Genet (2005) 70:116-130.

See Also

`find.haplo.beta.qt`, `haplo.em`, `haplo.power.cc`

Examples

```
haplo <- rbind(
               c(     1,    2,    2,    1,    2),
               c(     1,    2,    2,    1,    1),
               c(     1,    1,    2,    1,    1),
               c(     1,    2,    1,    1,    2),
               c(     1,    2,    2,    2,    1),
               c(     1,    2,    1,    1,    1),
               c(     1,    1,    2,    2,    1),
               c(     1,    1,    1,    1,    2),
               c(     1,    2,    1,    2,    1),
               c(     1,    1,    1,    2,    1),
               c(     2,    2,    1,    1,    2),
               c(     1,    1,    2,    1,    2),
               c(     1,    1,    2,    2,    2),
               c(     1,    2,    2,    2,    2),
               c(     2,    2,    2,    1,    2),
               c(     1,    1,    1,    1,    1),
               c(     2,    1,    1,    1,    1),
               c(     2,    1,    2,    1,    1),
               c(     2,    2,    1,    1,    1),
               c(     2,    2,    1,    2,    1),
               c(     2,    2,    2,    1,    1))
dimnames(haplo)[[2]] <- paste("loc", 1:ncol(haplo), sep=".")
haplo <- data.frame(haplo)

haplo.freq <- c(0.170020121, 0.162977867, 0.123742455, 0.117706237,
0.097585513, 0.084507042, 0.045271630, 0.039235412, 0.032193159,
0.019114688, 0.019114688, 0.013078471, 0.013078471, 0.013078471,
```

```
0.013078471, 0.006036217, 0.006036217, 0.006036217,
0.006036217, 0.006036217, 0.006036217)

# define index for risk haplotypes (having alleles 1-1 at loci 2 and 3)
haplo.risk <- (1:nrow(haplo))[haplo$loc.2==1 & haplo$loc.3==1]

# define index for baseline haplotype
base.index <-  1

# Because it can be easier to speficy genetic effect size in terms of
# a regression model R-squared value (r2), we use an
# auxiliary function to set up haplo.beta based on a specifed r2 value:

tmp <- find.haplo.beta.qt(haplo,haplo.freq,base.index,haplo.risk,
r2=0.01, y.mu=0, y.var=1)

haplo.beta <- tmp$beta

# Compute sample size for given power

haplo.power.qt(haplo, haplo.freq, base.index, haplo.beta, y.mu=0,
y.var=1, alpha=.05, power=.80)

# Compute power for given sample size

haplo.power.qt(haplo, haplo.freq, base.index, haplo.beta, y.mu=0,
y.var=1, alpha=.05, sample.size = 2091)
```

`haplo.scan` *Search for a trait-locus by sliding a fixed-width window over each marker locus and scanning all possible haplotype lengths within the window*

Description

Search for haplotypes that have the strongest association with a binary trait (typically case/control status) by sliding a fixed-width window over each marker locus and scanning all possible haplotype lengths within the window. For each haplotype length, a score statistic is computed to compare the set of haplotypes with a given length between cases versus controls. The locus-specific score statistic is the maximum score statistic calculated on loci containing that locus. The maximum score statistic over all haplotype lengths within all possible windows is used for a global test for association. Permutations of the trait are used to compute p-values.

Usage

```
haplo.scan(y, geno, width=4, miss.val=c(0, NA),
           em.control=haplo.em.control(),
           sim.control=score.sim.control())
```

```
haplo.scan.obs(y, em.obj, width)

haplo.scan.sim(y.reord, save.lst, nloci)
```

Arguments

y	Vector of binary trait values, must be 1 for cases and 0 for controls.
y.reord	Same as y, except the order is permuted
geno	Matrix of alleles, 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 ncol(geno) = 2*K. Rows represent alleles for each subject.
width	Width of sliding the window
miss.val	Vector of codes for missing values of alleles
em.control	A list of control parameters to determine how to perform the EM algorithm for estimating haplotype frequencies when phase is unknown. The list is created by the function haplo.em.control - see this function for more details.
sim.control	A list of control parameters to determine how simulations are performed for simulated p-values. The list is created by the function score.sim.control and the default values of this function can be changed as desired. See score.sim.control for details.
em.obj	Object returned from haplo.em, performed on geno
save.lst	Information on haplotypes needed for haplo.scan.sim, already calculated in haplo.scan
nloci	number of markers

Details

Search 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 considering all haplotype lengths within each window. To acount 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, 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. The global test statistic is the maximum over all computed score statistics. To compute p-values, the case/control status is randomly permuted. Simulations are performed until precision criteria are met for all p-values; the criteria are controlled by score.sim.control. See the note for long run times.

Value

A list that has class haplo.scan, which contains the following items:

call	The call to haplo.scan

scan.df	A data frame containing the maximum test statistic for each window around each locus, and its simulated p-value.
max.loc	The loci (locus) which contain(s) the maximum observed test statistic over all haplotype lengths and all windows.
globalp	A p-value for the significance of the global maximum statistic.
nsim	Number of simulations performed

Note

For datasets with many estimated haplotypes, the run-time can be very long.

References

Cheng et al-1 Cheng R, Ma JZ, Wright FA, Lin S, Gau X, Wang D, Elston RC, Li MD. "Nonparametric disequilibrium mapping of functional sites using haplotypes of multiple tightly linked single-nucleotide polymorphism markers". Genetics 164 (2003):1175-1187.

Cheng et al-2 Cheng R, Ma JZ, Elston RC, Li MD. "Fine Mapping Functional Sites or Regions from Case-Control Data Using Haplotypes of Multiple Linked SNPs." Annals of Human Genetics 69 (2005): 102-112.

See Also

haplo.em, haplo.em.control, score.sim.control

Examples

```
# create a random genotype matrix with 10 loci, 50 cases, 50 controls
set.seed(1)
tmp <- ifelse(runif(2000)>.3, 1, 2)
geno <- matrix(tmp, ncol=20)
y <- rep(c(0,1),c(50,50))

# search 10-locus region, typically don't limit the number of
# simulations, but run time can get long with many simulations

scan.obj <- haplo.scan(y, geno, width=3,
              sim.control = score.sim.control(min.sim=10, max.sim=20))

print(scan.obj)
```

---

haplo.score *Score Statistics for Association of Traits with Haplotypes*

Description

Compute score statistics to evaluate the association of a trait with haplotypes, when linkage phase is unknown and diploid marker phenotypes are observed among unrelated subjects. For now, only autosomal loci are considered.

Usage

```
haplo.score(y, geno, trait.type="gaussian", offset = NA, x.adj = NA,
            min.count=5, skip.haplo=min.count/(2*nrow(geno)),
            locus.label=NA, miss.val=c(0,NA), haplo.effect="additive",
            eps.svd=1e-5, simulate=FALSE, sim.control=score.sim.control(),
            em.control=haplo.em.control())
```

Arguments

y	Vector of trait values. For trait.type = "binomial", y must have values of 1 for event, 0 for no event.
geno	Matrix of alleles, 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 ncol(geno) = 2*K. Rows represent alleles for each subject.
trait.type	Character string defining type of trait, with values of "gaussian", "binomial", "poisson", "ordinal".
offset	Vector of offset when trait.type = "poisson"
x.adj	Matrix of non-genetic covariates used to adjust the score statistics. Note that intercept should not be included, as it will be added in this function.
min.count	The minimum number of counts for a haplotype to be included in the model. First, the haplotypes selected to score are chosen by minimum frequency greater than skip.haplo (based on min.count, by default). It is also used when haplo.effect is either dominant or recessive. This is explained best in the recessive instance, where only subjects who are homozygous for a haplotype will contribute information to the score for that haplotype. If fewer than min.count subjects are estimated to be affected by that haplotype, it is not scored. A warning is issued if no haplotypes can be scored.
skip.haplo	Minimum haplotype frequency for which haplotypes are scored in the model. By default, the frequency is based on "min.count" divided by the 2*N total haplotype occurrences in the sample.
locus.label	Vector of labels for loci, of length K (see definition of geno matrix)
miss.val	Vector of codes for missing values of alleles
haplo.effect	the "effect" of a haplotypes, which determines the covariate (x) coding of haplotypes. Valid options are "additive" (causing x = 0, 1, or 2, the count of a particular haplotype), "dominant" (causing x = 1 if heterozygous or homozygous carrier of a particular haplotype; x = 0 otherwise), and "recessive" (causing x = 1 if homozygous for a particular haplotype; x = 0 otherwise).
eps.svd	epsilon value for singular value cutoff; to be used in the generalized inverse calculation on the variance matrix of the score vector (see help(Ginv) for details).
simulate	Logical: if FALSE, no empirical p-values are computed; if TRUE, simulations are performed. Specific simulation parameters can be controlled in the sim.control parameter list.
sim.control	A list of control parameters to determine how simulations are performed for simulated p-values. The list is created by the function score.sim.control and the default values of this function can be changed as desired. See score.sim.control for details.

`em.control` A list of control parameters to determine how to perform the EM algorithm for estimating haplotype frequencies when phase is unknown. The list is created by the function haplo.em.control - see this function for more details.

Details

Compute the maximum likelihood estimates of the haplotype frequencies and the posterior probabilities of the pairs of haplotypes for each subject using an EM algorithm. The algorithm begins with haplotypes from a subset of the loci and progressively discards those with low frequency before inserting more loci. The process is repeated until haplotypes for all loci are established. The posterior probabilities are used to compute the score statistics for the association of (ambiguous) haplotypes with traits. The glm function is used to compute residuals of the regression of the trait on the non-genetic covariates.

Value

List with the following components:

`score.global` Global statistic to test association of trait with haplotypes that have frequencies >= skip.haplo.

`df` Degrees of freedom for score.global.

`score.global.p` P-value of score.global based on chi-square distribution, with degrees of freedom equal to df.

`score.global.p.sim` P-value of score.global based on simulations (set equal to NA when simulate=F).

`score.haplo` Vector of score statistics for individual haplotypes that have frequencies >= skip.haplo.

`score.haplo.p` Vector of p-values for score.haplo, based on a chi-square distribution with 1 df.

`score.haplo.p.sim` Vector of p-values for score.haplo, based on simulations (set equal to NA when simulate=F).

`score.max.p.sim` Simulated p-value indicating for simulations the number of times a maximum score.haplo value exceeds the maximum score.haplo from the original data (equal to NA when simulate=F).

`haplotype` Matrix of hapoltypes analyzed. The ith row of haplotype corresponds to the ith item of score.haplo, score.haplo.p, and score.haplo.p.sim.

`hap.prob` Vector of haplotype probabilies, corresponding to the haplotypes in the matrix haplotype.

`locus.label` Vector of labels for loci, of length K (same as input argument).

`call` The call to the haplo.score function; useful for recalling what parameters were used.

`haplo.effect` The haplotype effect model parameter that was selected for haplo.score.

`simulate` Same as function input parameter. If [T]rue, simulation results are included in the haplo.score object.

`n.val.global` Vector containing the number of valid simulations used in the global score statistic simulation. The number of valid simulations can be less than the number of simulations requested (by sim.control) if simulated data sets produce unstable variances of the score statistics.

`n.val.haplo` Vector containing the number of valid simulations used in the p-value simulations for maximum-score statistic and scores for the individual haplotypes.

References

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. "Score tests for association of traits with haplotypes when linkage phase is ambiguous." Amer J Hum Genet. 70 (2002): 425-434.

See Also

`haplo.em`, `plot.haplo.score`, `print.haplo.score`, `haplo.em.control`, `score.sim.control`

Examples

```
  # establish all hla.demo data,
  # remove genotypes with missing alleles just so haplo.score runs faster
  # with missing values included, this example takes 2-4 minutes
  # FOR REGULAR USAGE, DO NOT DISCARD GENOTYPES WITH MISSING VALUES

  data(hla.demo)
  geno <- as.matrix(hla.demo[,c(17,18,21:24)])
  keep <- !apply(is.na(geno) | geno==0, 1, any)
  hla.demo <- hla.demo[keep,]
  geno <- geno[keep,]
  attach(hla.demo)
  label <- c("DQB","DRB","B")

# For quantitative, normally distributed trait:

  score.gaus <- haplo.score(resp, geno, locus.label=label,
                            trait.type = "gaussian")
  print(score.gaus)

# For ordinal trait:
  y.ord <- as.numeric(resp.cat)
  score.ord <- haplo.score(y.ord, geno, locus.label=label,
                           trait.type="ordinal")
  print(score.ord)

# For a  binary trait and simulations,
# limit simulations to 500 in score.sim.control, default is 20000
  y.bin <-ifelse(y.ord==1,1,0)
  score.bin.sim <- haplo.score(y.bin, geno, trait.type = "binomial",
                    locus.label=label, simulate=TRUE,
                    sim.control=score.sim.control(min.sim=200,max.sim=500))

  print(score.bin.sim)
```

```
# For a binary trait, adjusted for sex and age:

  x <- cbind(male, age)
  score.bin.adj <- haplo.score(y.bin, geno, trait.type = "binomial",
                               locus.label=label, x.adj=x)
  print(score.bin.adj)
```

---

haplo.score.merge *Merge haplo.score And haplo.group Objects*

Description

Combine information from returned objects of haplo.score and haplo.group, 'score' and 'group' respectively. 'score' and 'group' are sorted differently and 'score' keeps a subset of all the haplotypes while 'group' has all of them. To combine results from the two objects, merge them by haplotype and sort by score of the haplotype. The merged object includes all haplotypes; i.e. those appearing in 'group', but the print default only shows haplotypes which have a score.

Usage

```
haplo.score.merge(score, group)
```

Arguments

score	Object returned from haplo.score of class "haplo.score".
group	Object returned from haplo.group of class "haplo.group".

Details

Haplo.score returns score statistic and p-value for haplotypes with an overall frequency above the user-specified threshold, skip.haplo. For haplotypes with frequencies below the threshold, the score and p-value will be NA. Overall haplotype frequencies and for sub-groups are estimated by haplo.group.

Value

Data frame including haplotypes, score-statistics, score p-value, estimated haplotype frequency for all subjects, and haplotype frequency from group subsets.

Side Effects

Warning: The merge will not detect if the group and score objects resulted from different subject phenotypes selected by memory-usage parameters, rm.geno.na and enum.limit. Users must use the same values for these parameters in haplo.score and haplo.group so the merged objects are consistent.

See Also

haplo.score, haplo.group

Examples

```
data(hla.demo)
geno <- as.matrix(hla.demo[,c(17,18,21:24)])
keep <- !apply(is.na(geno) | geno==0, 1, any)
hla.demo <- hla.demo[keep,]
geno <- geno[keep,]
attach(hla.demo)
y.ord <- as.numeric(resp.cat)
y.bin <-ifelse(y.ord==1,1,0)

group.bin <- haplo.group(y.bin, geno, miss.val=0)
score.bin <- haplo.score(y.bin, geno, trait.type="binomial")
score.merged <- haplo.score.merge(score.bin, group.bin)

print(score.merged)
```

---

haplo.score.slide *Score Statistics for Association of Traits with Haplotypes*

Description

Used to identify sub-haplotypes from a group of loci. Run haplo.score on all contiguous subsets of size n.slide from the loci in a genotype matrix (geno). From each call to haplo.score, report the global score statistic p-value. Can also report global and maximum score statistics simulated p-values.

Usage

```
haplo.score.slide(y, geno, trait.type="gaussian", n.slide=2,
                  offset = NA, x.adj = NA, min.count=5,
                  skip.haplo=min.count/(2*nrow(geno)),
                  locus.label=NA, miss.val=c(0,NA),
                  haplo.effect="additive", eps.svd=1e-5,
                  simulate=FALSE, sim.control=score.sim.control(),
                  em.control=haplo.em.control())
```

Arguments

y	Vector of trait values. For trait.type = "binomial", y must have values of 1 for event, 0 for no event.
geno	Matrix of alleles, 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 ncol(geno) = 2*K. Rows represent alleles for each subject.
trait.type	Character string defining type of trait, with values of "gaussian", "binomial", "poisson", "ordinal".

`n.slide` Number of loci in each contiguous subset. The first subset is the ordered loci numbered 1 to n.slide, the second subset is 2 through n.slide+1 and so on. If the total number of loci in geno is n.loci, then there are n.loci - n.slide + 1 total subsets.

`offset` Vector of offset when trait.type = "poisson"

`x.adj` Matrix of non-genetic covariates used to adjust the score statistics. Note that intercept should not be included, as it will be added in this function.

`min.count` The minimum number of counts for a haplotype to be included in the model. First, the haplotypes selected to score are chosen by minimum frequency greater than skip.haplo (based on min.count, by default). It is also used when haplo.effect is either dominant or recessive. This is explained best in the recessive instance, where only subjects who are homozygous for a haplotype will contribute information to the score for that haplotype. If fewer than min.count subjects are estimated to be affected by that haplotype, it is not scored. A warning is issued if no haplotypes can be scored.

`skip.haplo` For haplotypes with frequencies < skip.haplo, categorize them into a common group of rare haplotypes.

`locus.label` Vector of labels for loci, of length K (see definition of geno matrix).

`miss.val` Vector of codes for missing values of alleles.

`haplo.effect` The "effect" pattern of haplotypes on the response. This parameter determines the coding for scoring the haplotypes. Valid coding options for heterozygous and homozygous carriers of a haplotype are "additive" (1, 2, respectively), "dominant" (1,1, respectively), and "recessive" (0, 1, respectively).

`eps.svd` epsilon value for singular value cutoff; to be used in the generalized inverse calculation on the variance matrix of the score vector.

`simulate` Logical, if [F]alse (default) no empirical p-values are computed. If [T]rue simulations are performed. Specific simulation parameters can be controlled in the sim.control parameter list.

`sim.control` A list of control parameters used to perform simulations for simulated p-values in haplo.score. The list is created by the function score.sim.control and the default values of this function can be changed as desired.

`em.control` A list of control parameters used to perform the em algorithm for estimating haplotype frequencies when phase is unknown. The list is created by the function haplo.em.control and the default values of this function can be changed as desired.

Details

Haplo.score.slide is useful for a series of loci where little is known of the association between a trait and haplotypes. Using a range of n.slide values, the region with the strongest association will consistently have low p-values for locus subsets containing the associated haplotypes. The global p-value measures significance of the entire set of haplotypes for the locus subset. Simulated maximum score statistic p-values indicate when one or a few haplotypes are associated with the trait.

Value

List with the following components:

df	Data frame with start locus, global p-value, simulated global p-value, and simulated maximum-score p-value.
n.loci	Number of loci given in the genotype matrix.
simulate	Same as parameter description above.
haplo.effect	The haplotype effect model parameter that was selected for haplo.score.
n.slide	Same as parameter description above.
locus.label	Same as parameter description above.
n.val.haplo	Vector containing the number of valid simulations used in the maximum-score statistic p-value simulation. The number of valid simulations can be less than the number of simulations requested (by sim.control) if simulated data sets produce unstable variables of the score statistics.
n.val.global	Vector containing the number of valid simulations used in the global score statistic p-value simulation.

References

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. "Score tests for association of traits with haplotypes when linkage phase is ambiguous." Amer J Hum Genet. 70 (2002): 425-434.

See Also

haplo.score, plot.haplo.score.slide, score.sim.control

Examples

```
  data(hla.demo)

# Continuous trait slide by 2 loci on all 11 loci, uncomment to run it.
# Takes > 20 minutes to run
#  geno.11 <- hla.demo[,-c(1:4)]
#  label.11 <- c("DPB","DPA","DMA","DMB","TAP1","TAP2","DQB","DQA","DRB","B","A")
#  slide.gaus <- haplo.score.slide(hla.demo$resp, geno.11, trait.type = "gaussian",
#                                  locus.label=label.11, n.slide=2)

#  print(slide.gaus)
#  plot(slide.gaus)

# Run shortened example on 9 loci
# For an ordinal trait, slide by 3 loci, and simulate p-values:
#  geno.9 <- hla.demo[,-c(1:6,15,16)]
#  label.9 <- c("DPA","DMA","DMB","TAP1","DQB","DQA","DRB","B","A")

#  y.ord <- as.numeric(hla.demo$resp.cat)

# data is set up, to run, run these lines of code on the data that was
```

```
# set up in this example. It takes > 15 minutes to run
#  slide.ord.sim <-  haplo.score.slide(y.ord, geno.9, trait.type = "ordinal",
#                      n.slide=3, locus.label=label.9, simulate=TRUE,
#                      sim.control=score.sim.control(min.sim=200, max.sim=500))

  # note, results will vary due to simulations
#  print(slide.ord.sim)
#  plot(slide.ord.sim)
#  plot(slide.ord.sim, pval="global.sim")
#  plot(slide.ord.sim, pval="max.sim")
```

`hapPower.demo` *Set of haplotypes and frequencies for power and sample size calculations*

Description

An example set of haplotypes and frequencies for power and sample size calculations in haplo.power.cc and haplo.power.qt

Usage

```
data(hapPower.demo)
```

Format

A data frame with 21 observations on the following 6 variables.

`loc.1` allele 1 in the haplotype

`loc.2` allele 2 in the haplotype

`loc.3` allele 3 in the haplotype

`loc.4` allele 4 in the haplotype

`loc.5` allele 5 in the haplotype

`haplo.freq` numeric, frequency of haplotype

References

Schaid, DJ. Power and sample size for testing associations of haplotypes with complex traits. *Ann Hum Genet* (2005) 70:116-130.

Examples

```
data(hapPower.demo)
```

`hla.demo` *HLA Loci and Serologic Response to Measles Vaccination*

Description

A data frame with genotypes at eleven HLA-region loci genotyped for 220 subjects, phase not known. Contains measles vaccination response with covariate data.

Usage

```
data(hla.demo)
```

Format

A data frame with 220 observations on the following 26 variables.

`resp` numeric, Quantitative response to Measles Vaccination

`resp.cat` Category of vaccination response, a factor with levels `high` `low` `normal`

`male` numeric, indicator of gener, 1=male, 0=female

`age` numeric, subject's age

`DPB.a1` first allele of genotype

`DPB.a2` second allele of genotype

`DPA.a1` first allele of genotype

`DPA.a2` second allele of genotype

`DMA.a1` first allele of genotype

`DMA.a2` second allele of genotype

`DMB.a1` first allele of genotype

`DMB.a2` second allele of genotype

`TAP1.a1` first allele of genotype

`TAP1.a2` second allele of genotype

`TAP2.a1` first allele of genotype

`TAP2.a2` second allele of genotype

`DQB.a1` first allele of genotype

`DQB.a2` second allele of genotype

`DQA.a1` first allele of genotype

`DQA.a2` second allele of genotype

`DRB.a1` first allele of genotype

`DRB.a2` second allele of genotype

`B.a1` first allele of genotype

`B.a2` second allele of genotype

`A.a1` first allele of genotype

`A.a2` second allele of genotype

Source

Data set kindly provided by Gregory A. Poland, M.D. and the Mayo Clinic Vaccine Research Group for illustration only, and my not be used for publication.

References

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. "Score tests for association of traits with haplotypes when linkage phase is ambiguous." Amer J Hum Genet. 70 (2002): 425-434.

Examples

```
data(hla.demo)
```

---

locator.haplo *Find Location from Mouse Clicks and Print Haplotypes on Plot*

Description

Much like the R/Splus locator function is used to find x-y coordinates on a plot. Find all x-y coordinates that are chosen by the user's mouse clicks. Then print haplotype labels at the chosen positions.

Usage

```
locator.haplo(obj)
```

Arguments

obj	An object (of class haplo.score) that is returned from haplo.score.

Details

After plotting the results in obj, as from plot(obj), the function locator.haplo is used to place on the plot the text strings for haplotypes of interest. After the function call (e.g., locator.haplo(obj)), the user can click, with the left mouse button, on as many points in the plot as desired. Then, clicking with the middle mouse button will cause the haplotypes to be printed on the plot. The format of a haplotype is "a:b:c", where a, b, and c are alleles, and the separator ":" is used to separate alleles on a haplotype. The algorithm chooses the closest point that the user clicks on, and prints the haplotype either above the point (for points on the lower-half of the plot) or below the point (for points in the upper-half of the plot).

Value

List with the following components:

x.coord	Vector of x-coordinates.
y.coord	Vector of y-coordinates.
hap.txt	Vector of character strings for haplotypes.

See Also

haplo.score

Examples

```
# follow the pseudo-code
#  score.out <-  haplo.score(y, geno, trait.type = "gaussian")

#  plot(score.out)

#  locator.haplo(score.out)
```

---

locus *Creates an object of class "locus"*

Description

Creates an object containing genotypes for multiple individuals. The object can then use method functions developed for objects of class "locus".

Usage

```
locus(allele1, allele2, chrom.label=NULL,locus.alias=NULL,
     x.linked=FALSE, sex=NULL, male.code="M", female.code="F", miss.val=NA)
```

Arguments

allele1	A vector containing the labels for 1 allele for a set of individuals, or optionally a matrix with 2 columns each containing an allele for each person.
allele2	A vector containing the labels for the second allele for a set of individuals. If allele 1 is a matrix, allele 2 need not be specified.
chrom.label	A label describing the chromosome the alleles belong to
locus.alias	A vector containing one or more aliases describing the locus. The first alias in the vector will be used as a label for printing in some functions such as multilocus.print().
x.linked	A logical value denoting whether the chromosome is x linked
sex	A vector containing the gender of each individual (required if x.linked=T)
male.code	The code denoting a male in the sex vector
female.code	The code denoting a female in the sex vector
miss.val	a vector of codes denoting missing values for allele1 and allele2. Note that NA will always be treated as a missing value, even if not specified in miss.val. Also note that if multiple missing value codes are specified, the original missing value code for a specific individual can not be retrieved from the locus object.

Value

Returns an object of class locus which inherits from class model.matrix containing the following elements:

geno	a matrix with 2 columns where each row contains numeric codes for the 2 alleles for an individual.
chrom.label	a chromosome label
locus.alias	a vector of aliases for the locus
x.linked	a logical value specifying if the locus is x-linked or not
allele.labels	a vector of labels corresponding to the numeric codes in matrix geno (similar to levels in a factor)
male.code	a code to be used to identify males for an x.linked locus.
female.code	a code to be used to identify females for an x.linked locus.

Examples

```
b1 <- c("A","A","B","C","E","D")
b2 <- c("A","A","C","E","F","G")
loc1 <- locus(b1,b2,chrom=4,locus.alias="D4S1111")

loc1

# a second example which uses more parameters, some may not be supported.
c1 <- c(101,10, 112,112,21,112)
c2 <- c(101,101,112,  100,21, 10)

gender <- rep(c("M","F"),3)
loc2 <- locus(c1,c2,chrom="X",locus.alias="DXS1234", x.linked=TRUE, sex=gender)

loc2
```

---

louis.info *Louis Information for haplo.glm*

Description

For internal use within the haplo.stats library's haplo.glm function

Usage

```
louis.info(fit, epsilon=1e-8)
```

Arguments

fit	glm fitted object
epsilon	cut-off for singular values in the generalized inverse of the information matrix

na.geno.keep *Remove rows with NA in covariates, but keep genotypes with NAs*

Description

Removes rows with NA in response or covariates, but keeps subjects with NAs in their genotypes if not missing all alleles.

Usage

```
na.geno.keep(m)
```

Arguments

m	model matrix

Value

a model matrix with rows removed if exclusion criteria requires it

plot.haplo.score *Plot Haplotype Frequencies versus Haplotype Score Statistics*

Description

Method function to plot a class of type haplo.score

Usage

```
## S3 method for class 'haplo.score'
plot(x, ...)
```

Arguments

x	The object returned from haplo.score (which has class haplo.score).
...	Dynamic parameter for the values of additional parameters for the plot method.

Details

This is a plot method function used to plot haplotype frequencies on the x-axis and haplotype-specific scores on the y-axis. Because haplo.score is a class, the generic plot function can be used, which in turn calls this plot.haplo.score function.

Value

Nothing is returned.

References

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. "Score tests for association of traits with haplotypes when linkage phase is ambiguous." Amer J Hum Genet. 70 (2002): 425-434.

See Also

haplo.score

Examples

```
  data(hla.demo)
  geno <- as.matrix(hla.demo[,c(17,18,21:24)])
  keep <- !apply(is.na(geno) | geno==0, 1, any)
  hla.demo <- hla.demo[keep,]
  geno <- geno[keep,]
  attach(hla.demo)
  label <- c("DQB","DRB","B")

# For quantitative, normally distributed trait:

  score.gaus <- haplo.score(resp, geno, locus.label=label,
                            trait.type = "gaussian")

  plot.haplo.score(score.gaus)
  ## try: locator.haplo(1)
```

---

plot.haplo.score.slide

Plot a haplo.score.slide Object

Description

Method function to plot an object of class haplo.score.slide. The p-values from haplo.score.slide are for sub-haplotypes of a larger chromosomal region, and these are plotted to visualize the change in p-values as the sub-haplotype "slides" over a chromosome. Plot -log10(p-value) on the y-axis vs. the loci over which it was computed on the x-axis.

Usage

```
## S3 method for class 'haplo.score.slide'
plot(x, pval="global", dist.vec=1:x$n.loci, ...)
```

Arguments

x	The object returned from haplo.score.slide
pval	Character string for the choice of p-value to plot. Options are: "global" (the global score statistic p-value based on an asymptotic chi-square distribution), "global.sim" (the global score statistic simulated p-value), and "max.sim" (the simulated p-value for the maximum score statistic).

dist.vec	Numeric vector for position (i.e., in cM) of the loci along a chromosome. Distances on x-axis will correspond to these positions.
...	Dynamic parameter for the values of additional parameters for the plot method. Some useful options for manageing the x-axis labels are cex.axis, las, and srt.

Details

The x-axis has tick marks for all loci. The y-axis is the -log10() of the selected p-value. For each haplo.score result, plot a horizontal line at the height of -log10(p-value) drawn across the loci over which it was calculated. Therefore a p-value of 0.001 for the first 3 loci will plot as a horizontal line plotted at y=3 covering the first three tick marks. If the p-value for a set of loci is zero or very near zero, it is set to a minimum. Global asymptotic p-values of zero are set to the minimum of an epsilon or the lowest non-zero p-value in the region. Simulated p-values equal to zero are set to 0.5 divided by the total number of simulations performed.

Value

Nothing is returned.

References

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. "Score tests for association of traits with haplotypes when linkage phase is ambiguous." Amer J Hum Genet. 70 (2002): 425-434.

See Also

haplo.score.slide

Examples

```
# This example has a long run-time, therefore it is commented

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

#For an ordinal trait, slide by 3 loci, and simulate p-values:
#   y.ord <- as.numeric(resp.cat)
#   slide.ord.sim <-  haplo.score.slide(y.ord, geno.11, trait.type = "ordinal",
#                              n.slide=3, locus.label=label.11, simulate=TRUE,
#                              sim.control=score.sim.control(min.sim=500))

#   print(slide.ord.sim)
#   plot(slide.ord.sim)
#   plot(slide.ord.sim, pval="global.sim", las=2, cex.axis=.8)
#   plot(slide.ord.sim, pval="max.sim", srt=90, cex.axis=.8)
```

`plot.seqhap` *Plot a seqhap object*

Description

Method to plot an object of class seqhap. The p-values at each locus are based on sequentially combined loci, and they are plotted to visualize the p-values when scanning each locus using seqhap methods. Plots -log10(p-value) on the y-axis vs. the loci over which it was computed on the x-axis.

Usage

```
## S3 method for class 'seqhap'
plot(x, pval="hap", single=TRUE,
minp=.Machine$double.eps, ...)
```

Arguments

`x`	The object returned from seqhap
`pval`	Character string for the choice of p-value to plot. Options are: "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).
`single`	Logical, indicating whether to plot p-values for single-locus association tests. If TRUE, the pointwise p-values from the single-locus will be plotted using a dotted line.
`minp`	Smallest "allowable" p-value; any p-value smaller will be set to log10(minp). The default is the value closest to zero that can be represented in Splus/R.
`...`	Dynamic parameter for the values of additional parameters for the plot method. Accept the ylim parameter for plot() and other parameters for lines(), points(), and axis(). Recommended values to make locus labels vertical on the x-axis: for R: las=2, cex.axis=1.2 for S+: srt=90, cex.axis=1.2, adj=1

Details

The x-axis has tick marks for all loci. The y-axis is the -log10() of the selected p-value. 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.

If the permutation p-value is zero, for plotting purposes it is set to 1/(n.sim+1).

Value

Nothing is returned.

References

Yu Z, Schaid DJ. (2007) Sequential haplotype scan methods for association analysis. Genet Epidemiol, in print.

See Also

seqhap, print.seqhap

Examples

```
  ## Not run:
    data(seqhap.dat)
    mydata.y <- seqhap.dat[,1]
    mydata.x <- seqhap.dat[,-1]
    data(seqhap.pos)
    myobj <- seqhap(y=mydata.y, geno=mydata.x, pos=seqhap.pos$pos)
    plot(myobj)

## End(Not run)
```

---

print.haplo.cc *Print a haplo.cc object*

Description

Display results for a haplotype analysis on a case-control study.

Usage

```
## S3 method for class 'haplo.cc'
print(x, order.by=c("score","haplotype","freq"), digits=max(options()$digits-2, 5),
    nlines=NULL, ...)
```

Arguments

x	A haplo.cc object, made by the haplo.cc function.
order.by	Order the printed data frame by the column: haplotype score (score), haplotype alleles (haplotype), or haplotype frequency (freq).
digits	Number of digits to display for the numeric columns of the data frame.
nlines	Print the first nlines of the cc.df data frame of the haplo.cc object, keeps output short if desired.
...	Dynamic parameter for the values of additional parameters for the print method.

Value

Nothing is returned.

See Also

haplo.cc

Examples

```
## for a haplo.cc object named cc.test,
##    order results by haplotype
# print.haplo.cc(cc.test, order.by="haplotype")
```

`print.haplo.em` *Print contents of a haplo.em object*

Description

Print a data frame with haplotypes and their frequencies. Likelihood information is also printed.

Usage

```
## S3 method for class 'haplo.em'
print(x, digits=max(options()$digits-2, 5), nlines=NULL, ...)
```

Arguments

`x`	A haplo.em object
`digits`	number of significant digits to print for numeric values
`nlines`	To shorten output, print the first 1:nlines rows of the large data frame.
`...`	optional arguments for print

Value

Nothing is returned

See Also

haplo.em

`print.haplo.group` *Print a haplo.group object*

Description

Method function to print a class of type haplo.group

Usage

```
## S3 method for class 'haplo.group'
print(x, digits=max(options()$digits-2, 5), nlines=NULL, ...)
```

Arguments

`x`	The object returned from haplo.group (which has old class haplo.group).
`digits`	Set the number of significant digits to print for haplotype probabilities.
`nlines`	For shorter output, print first 1:nlines rows of the large data frame
`...`	Optional arguments for the print method

Details

This is a print method function used to print information from the haplo.group class, with haplotype-specific information given in a table. Because haplo.group is a class, the generic print function can be used, which in turn calls this print.haplo.group function.

Value

Nothing is returned.

References

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. Expected haplotype frequencies for association of traits with haplotypes when linkage phase is ambiguous. Submitted to Amer J Hum Genet.

See Also

haplo.score, haplo.group, haplo.em

print.haplo.scan *Print a haplo.scan object*

Description

Print a haplo.scan object

Usage

```
## S3 method for class 'haplo.scan'
print(x, digits=max(options()$digits - 2, 5), ...)
```

Arguments

x	An object created by haplo.scan
digits	Significant digits shown for numeric data
...	Options parameters for the print function

Value

NULL

See Also

haplo.scan

print.haplo.score *Print a haplo.score object*

Description

Method function to print a class of type haplo.score

Usage

```
## S3 method for class 'haplo.score'
print(x, digits, nlines=NULL, ...)
```

Arguments

x	The object returned from haplo.score (which has class haplo.score).
digits	Number of digits to round the numeric output.
nlines	Print the first 'nlines' rows of the large data frame for fast, short view of the results.
...	Dynamic parameter for the values of additional parameters for the print method.

Details

This is a print method function used to print information from haplo.score class, with haplotype-specific information given in a table. Because haplo.score is a class, the generic print function can be used, which in turn calls this print.haplo.score function.

Value

If print is assigned, the object contains the table of haplotype scores that was printed by the method

See Also

haplo.score

---

print.haplo.score.merge

Print a haplo.score.merge object

Description

Method function to print a class of type haplo.score.merge

Usage

```
## S3 method for class 'haplo.score.merge'
print(x, order.by="score", all.haps=FALSE,
                digits=max(options()$digits-2, 5), nlines=NULL, ...)
```

Arguments

x	The object returned from haplo.score.merge (which has old class {S} haplo.score.merge).
order.by	Column of the haplo.score.merge object by which to order the results
all.haps	Logical, if (T)rue prints a row for all haplotypes. If (F)alse, the default, only prints the haplotypes kept in haplo.score for modelling.
digits	Set the number of significant digits to print for the numeric output.
nlines	Print the first 'nlines' rows of the large data frame for a short view of the results.
...	Dynamic parameter for the values of additional parameters for the print method.

Details

This is a print method function used to print information from the haplo.score.merge class. Because haplo.score.merge is a class, the generic print function can be used, which in turn calls this print.haplo.score.merge function.

Value

Nothing is returned.

References

Schaid DJ, Rowland CM, Tines DE, Jacobson RM, Poland GA. Expected haplotype frequencies for association of traits with haplotypes when linkage phase is ambiguous. Submitted to Amer J Hum Genet.

See Also

haplo.score.merge, haplo.score, haplo.group

Examples

```
#see example for haplo.score.merge
```

---

print.haplo.score.slide

Print the contents of a haplo.score.slide object

Description

Print the data frame returned from haplo.score.slide

Usage

```
## S3 method for class 'haplo.score.slide'
print(x, digits=max(options()$digits - 2, 5), ...)
```

Arguments

x	A haplo.score.slide object
digits	Number of digits to print for numeric output
...	Optional arguments for the print method

---

printBanner

Print a nice banner

Description

Print a centered banner that carries to multiple lines

Usage

```
printBanner(str, banner.width=options()$width, char.perline=.75*banner.width, border="=")
```

Arguments

str	character string - a title within the banner
banner.width	width of banner, the default is set to fit current options
char.perline	number of characters per line for the title, the default is 75% of the banner.width parameter
border	type of character for the border

Details

This function prints a nice banner in both R and S-PLUS

Value

nothing is returned

See Also

options

Examples

```
printBanner("This is a pretty banner", banner.width=40, char.perline=30)

# the output looks like this:
# ========================================
#         This is a pretty banner
# ========================================
```

---

residuals.haplo.glm *Accessing residuals for haplo.glm fit*

Description

Access the residuals from a haplo.glm model fit

Usage

```
## S3 method for class 'haplo.glm'
residuals(object, type=c("deviance", "pearson",
                                  "working", "response"), ...)
```

Arguments

object	A haplo.glm object
type	Type of residuals to return. Options are "deviance" (default), "pearson", "working", and "response". Partial residuals not supported in this method.
...	Optional arguments

Details

Many of the subjects in a haplo.glm fit are expanded in the model matrix with weights used to reflect the posterior probability of the subject's haplotype pairs given their genotype. The working residuals within the fitted object are from this expanded model matrix, and the residuals in this method are calculated from the weighted fitted value for the subject across all their haplotype pairs.

Value

Residuals for each person in the model.

See Also

haplo.glm, residuals.glm, fitted.haplo.glm

---

score.sim.control *Create the list of control parameters for simulations in haplo.score*

Description

In the call to haplo.score, the sim.control parameter is a list of parameters that control the simulations. This list is created by this function, score.sim.control, making it easy to change the default values.

Usage

```
score.sim.control(p.threshold=0.25, min.sim=1000, max.sim=20000.,verbose=FALSE)
```

Arguments

p.threshold	A paremeter used to determine p-value precision from Besag and Clifford (1991). For a p-value calculated after min.sim simulations, continue doing simulations until the p-value's sample standard error is less than p.threshold * p-value. The dafault value for p.threshold = 1/4 corresponds approximately to having a two-sided 95% confidence interval for the p-value with a width as wide as the p-value itself. Therefore, simulations are more precise for smaller p-values. Additionally, since simulations are stopped as soon as this criteria is met, p-values may be biased high.
min.sim	The minimum number of simulations to run. To run exactly min.sim simulations, set max.sim = min.sim. Also, if run-time is an issue, a lower minimum (e.g. 500) may be useful, especially when doing simulations in haplo.score.slide.
max.sim	The upper limit of simulations allowed. When the number of simulations reaches max.sim, p-values are approximated based on simulation results at that time.
verbose	Logical, if (T)rue, print updates from every simulation to the screen. If (F)alse, do not print these details.

Details

In simulations for haplo.score, employ the simulation p-value precision criteria of Besag and Clifford (1991). The criteria ensures both the global and the maximum score statistic simulated p-values be precise for small p-values. First, perform min.sim simulations to guarantee sufficient precision for the score statistics on individual haplotypes. Then continue simulations as needed until simulated p-values for both the global and max score statistics meet precision requirements set by p.threshold.

Value

A list of the control parameters:

`p.threshold`	As described above
`min.sim`	As described above.
`max.sim`	As described above
`verbose`	As described above

References

Besag, J and Clifford, P. "Sequential Monte Carlo p-values." Biometrika. 78, no. 2 (1991): 301-304.

See Also

`haplo.score`

Examples

```
# it would be used in haplo.score as appears below
#
# score.sim.500 <- haplo.score(y, geno, trait.type="gaussian", simulate=T,
#                sim.control=score.sim.control(min.sim=500, max.sim=2000)
```

---

seqhap *Sequential Haplotype Scan Association Analysis for Case-Control Data*

Description

Seqhap implements sequential haplotype scan methods to perform association analyses for case-control data. When evaluating each locus, loci that contribute additional information to haplotype associations with disease status will be added sequentially. This conditional evaluation is based on the Mantel-Haenszel (MH) test. Two sequential methods are provided, a sequential haplotype method and a sequential summary method, as well as results based on the traditional single-locus method. Currently, seqhap only works with bialleleic loci (single nucleotide polymorphisms, or SNPs) and binary traits.

Usage

```
seqhap(y, geno, pos, locus.label=NA, weight=NULL,
       mh.threshold=3.84, r2.threshold=0.95, haplo.freq.min=0.005,
       miss.val=c(0, NA), sim.control=score.sim.control(),
       control=haplo.em.control())
## S3 method for class 'seqhap'
print(x, digits=max(options()$digits-2, 5), ...)
```

Arguments

y	vector of binary response (1=case, 0=control). The length is equal to the number of rows in geno.
geno	matrix of alleles, 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 ncol(geno)=2*K. Rows represent the alleles for each subject. Currently, only bi-allelic loci (SNPs) are allowed.
pos	vector of physical positions (or relative physical positions) for loci. If there are K loci, length(pos)=K. The scale (in kb, bp, or etc.) doesn't affect the results.
locus.label	vector of labels for the set of loci
weight	weights for observations (rows of geno matrix).
mh.threshold	threshold for the Mantel-Haenszel statistic that evaluates whether a locus contributes additional information of haplotype association to disease, conditional on current haplotypes. The default is 3.84, which is the 95th percentile of the chi-square distribution with 1 degree of freedom.
r2.threshold	threshold for a locus to be skipped. When scanning locus k, loci with correlations r-squared (the square of the Pearson's correlation) greater than r2.threshold with locus k will be ignored, so that the haplotype growing process continues for markers that are further away from locus k.
haplo.freq.min	the minimum haplotype frequency for a haplotype to be included in the association tests. The haplotype frequency is based on the EM algorithm that estimates haplotype frequencies independent of trait.
miss.val	vector of values that represent missing alleles.
sim.control	A list of control parameters to determine how simulations are performed for permutation p-values, similar to the strategy in haplo.score. The list is created by the function score.sim.control and the default values of this function can be changed as desired. Permutations are performed until a p.threshold accuracy rate is met for the three region-based p-values calculated in seqhap. See score.sim.control for details.
control	A list of parameters that control the EM algorithm for estimating haplotype frequencies when phase is unknown. The list is created by the function haplo.em.control - see this function for more details.
x	a seqhap object to print
digits	Number of significant digits to print for numeric values
...	Additional parameters for the print method

Details

No further details

Value

list with components:

converge	indicator of convergence of the EM algorithm (see haplo.em); 1 = converge, 0=failed
locus.label	vector of labels for loci
pos	chromosome positions for loci, same as input.
n.sim	number of permutations performed for emperical p-values
inlist	matrix that shows which loci are combined for association analysis in the sequential scan. The non-zero values of the kth row of inlist are the indices of the loci combined when scanning locus k.
chi.stat	chi-square statistics of single-locus analysis.
chi.p.point	permuted pointwise p-values of single-locus analysis.
chi.p.region	permuted regional p-value of single-locus analysis.
hap.stat	chi-square statistics of sequential haplotype analysis.
hap.df	degrees of freedom of sequential haplotype analysis.
hap.p.point	permuted pointwise p-values of sequential haplotype analysis.
hap.p.region	permuted region p-value of sequential haplotype analysis.
sum.stat	chi-square statistics of sequential summary analysis.
sum.df	degrees of freedom of sequential summary analysis.
sum.p.point	permuted pointwise p-values of sequential summary analysis.
sum.p.region	permuted regional p-value of sequential summary analysis.

References

Yu Z, Schaid DJ. (2007) Sequential haplotype scan methods for association analysis. Genet Epidemiol, in print.

See Also

haplo.em, print.seqhap, plot.seqhap, score.sim.control

Examples

```
# load example data with response and genotypes.
data(seqhap.dat)
mydata.y <- seqhap.dat[,1]
mydata.x <- seqhap.dat[,-1]
# load positions
data(seqhap.pos)
pos <- seqhap.pos$pos
```

```
# run seqhap with default settings
## Not run:
  # this example takes 5-10 seconds to run
  myobj <- seqhap(y=mydata.y, geno=mydata.x, pos=pos)
  print.seqhap(myobj)

## End(Not run)
```

`seqhap.dat` *Simulated data for seqhap examples*

Description

Simulated data set for the demonstration of seqhap functionality. Contains one column for disease status and columns representing 10 SNP loci with a known association. seqhap.pos contains a column for chromosome position, as required by seqhap.

Usage

```
data(seqhap.dat)
data(seqhap.pos)
```

Format

A data frame with 1000 observations on the following 21 variables.

`disease` numeric, indicator of disease status 0=no, 1=yes

`m1.1` first allele of genotype

`m1.2` second allele of genotype

`m2.1` first allele of genotype

`m2.2` second allele of genotype

`m3.1` first allele of genotype

`m3.2` second allele of genotype

`m4.1` first allele of genotype

`m4.2` second allele of genotype

`m5.1` first allele of genotype

`m5.2` second allele of genotype

`m6.1` first allele of genotype

`m6.2` second allele of genotype

`m7.1` first allele of genotype

`m7.2` second allele of genotype

`m8.1` first allele of genotype

`m8.2` second allele of genotype

`m9.1` first allele of genotype

`m9.2` second allele of genotype

`m10.1` first allele of genotype

`m10.2` second allele of genotype

References

Yu Z, Schaid DJ (2007) Sequantial haplotype scan methods for association analysis. *Gen Epi*, in print.

Examples

```
data(seqhap.dat)
```

`setupGeno` *Create a group of locus objects from a genotype matrix, assign to 'model.matrix' class.*

Description

The function makes each pair of columns a locus object, which recodes alleles to numeric and saves the original alleles as an attribute of the model.matrix.

Usage

```
setupGeno(geno, miss.val=c(0,NA), locus.label=NULL)
```

Arguments

`geno`	Matrix of alleles, 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 ncol(geno) = 2*K. Rows represent alleles for each subject.
`miss.val`	A vector of codes denoting missing values for allele1 and allele2. Note that NA will always be treated as a missing value, even if not specified in miss.val. Also note that if multiple missing value codes are specified, the original missing value code for a specific individual can not be retrieved from the loci object.
`locus.label`	vector of labels for the loci

Details

This function contains the essential parts of the loci function, which is no longer within haplo.stats

Value

A 'model.matrix' object with the alleles recoded to numeric values, and the original values are stored in the 'unique.alleles' attribute. The ith item of the unique.alleles list is a vector of unique alleles for the ith locus.

Note

A matrix that contains all elements of mode character will be sorted in alphabetic order. This order may differ across platforms according to your setting of LC_COLLATE. See the note in `haplo.em` about how this sort order affects results.

See Also

`locus`, `haplo.glm`, `haplo.em`

Examples

```
# Create some loci to work with
a1 <- 1:6
a2 <- 7:12

b1 <- c("A","A","B","C","E","D")
b2 <-c("A","A","C","E","F","G")

c1 <- c("101","10","115","132","21","112")
c2 <- c("100","101","0","100","21","110")

myGeno <- data.frame(a1,a2,b1,b2,c1,c2)
myGeno <- setupGeno(myGeno)
myGeno

attributes(myGeno)$unique.alleles
```

---

summary.haplo.em *Summarize contents of a haplo.em object*

Description

Display haplotype pairs and their posterior probabilities by subject. Also display a table with number of max haplotype pairs for a subject versus how many were kept (max vs. used).

Usage

```
## S3 method for class 'haplo.em'
summary(object, show.haplo=FALSE, digits=max(options()$digits-2, 5), nlines=NULL, ...)
```

Arguments

`object`	A haplo.em object
`show.haplo`	Logical. If TRUE, show the alleles of the haplotype pairs, otherwise show only the recoded values.
`digits`	number of significant digits to be printed for numeric values
`nlines`	To shorten output, print the first 1:nlines rows of the large data frame.
`...`	Optional arguments for the summary method

Value

A data.frame with a row for every subject's possible haplotype pairs and the posterior probabilities of that pair given their genotypes.

See Also

haplo.em

Examples

```
data(hla.demo)
geno <- hla.demo[,c(17,18,21:24)]
label <-c("DQB","DRB","B")
keep <- !apply(is.na(geno) | geno==0, 1, any)

save.em.keep <- haplo.em(geno=geno[keep,], locus.label=label)
save.df <- summary(save.em.keep)
save.df[1:10,]
```

---

summary.haplo.glm *Print and summary of a haplo.glm object*

Description

Do print and summary as in regular glm, then display extra information on haplotypes used in the model fit

Usage

```
## S3 method for class 'haplo.glm'
summary(object, show.all.haplo=FALSE,
                            show.missing=FALSE, ...)
## S3 method for class 'summary.haplo.glm'
print(x, digits = max(getOption("digits")-3,3), ...)
```

Arguments

x	A haplo.glm object
object	A haplo.glm object
show.all.haplo	Logical. If TRUE, print all haplotypes considered in the model.
show.missing	Logical. If TRUE, print number of rows removed because of missing values (NA) in y or x-covariates, or all alleles missing in geno
digits	Number of numeric digits to print.
...	Optional arguments for summary method

Details

Uses print.glm for the first section, then prints information on the haplotypes.

Value

If print is assigned, the object contains a list with the coefficient and haplotype data.frames which are printed by the method.

See Also

haplo.glm

---

summaryGeno *Summarize Full Haplotype Enumeration on Genotype Matrix*

Description

Provide a summary of missing allele information for each individual in the genotype matrix. The number of loci missing zero, one, or two alleles is computed, as well as the total number of haplotype pairs that could result from the observed phenotype.

Usage

```
summaryGeno(geno, miss.val=0)
```

Arguments

geno	Matrix of alleles, 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 geno has 2*K columns. Rows represent all observed alleles for each subject.
miss.val	Vector of codes for allele missing values.

Details

After getting information on the individual loci, this function makes a call to geno.count.pairs(). The E-M steps to estimate haplotype frequencies considers haplotypes that could result from a phenotype with a missing allele. It will not remove a subject's phenotype, only the unlikely haplotypes that result from it.

Value

Data frame with columns representing the number of loci with zero, one, and two missing alleles, then the total haplotype pairs resulting from full enumeration of the phenotype.

See Also

geno.count.pairs, haplo.em

vcov.haplo.glm *variance-covariance matrix of a fitted haplo.glm object*

Description

Returns the variance-covariance matrix of the main parameters of a fitted haplo.glm object

Usage

```
## S3 method for class 'haplo.glm'
vcov(object, freq=TRUE, ...)
```

Arguments

object	A haplo.glm object
freq	Logical. If TRUE, return the full covariance matrix including the entries for the frequency parameters
...	Optional arguments for print method

Details

var.mat is pre-computed in haplo.glm, the generalized inverse of the Louis information matrix

Value

Variance-covariance matrix of model parameters

See Also

haplo.glm

x.sexcheck *consistency checks for x.linked locus*

Description

Given an x.linked locus object and a vector of gender codes, the function will check to make sure the gender codes match the codes used to originally define the locus, and that no individuals defined as males are heterozygous.

Usage

```
x.sexcheck(x, sex, stop=FALSE)
```

Arguments

x	an object of class locus
sex	a vector of codes identifying the gender of each individual contained in the locus object
stop	if T , any warnings are converted to errors and execution is halted immediately

Value

T if one or more errors were found F if no errors were found

See Also

locus

Examples

```
c1 <- c(101,10, 112,112,21,112)
c2 <- c(101,101,112,100,21, 10)

gender <- rep(c("M","F"),3)
loc2 <- locus(c1,c2,chrom="X",locus.alias="DXS1234", x.linked=TRUE, sex=gender)

loc2
```

Index

∗ **classes**
  locus, 48
∗ **datasets**
  hapPower.demo, 45
  hla.demo, 46
  seqhap.dat, 65
∗ **design**
  haplo.power.cc, 30
  haplo.power.qt, 32
∗ **glm**
  anova.haplo.glm, 3
  fitted.haplo.glm, 7
  haplo.glm, 20
  haplo.glm.control, 25
  na.geno.keep, 50
  residuals.haplo.glm, 60
  summary.haplo.glm, 68
  vcov.haplo.glm, 70
∗ **matrix**
  Ginv, 11
∗ **models**
  haplo.design, 14
  haplo.model.frame, 29
∗ **power**
  chisq.power, 4
  f.power, 5
  find.haplo.beta.qt, 6
∗ **scores**
  haplo.score, 37
  haplo.score.merge, 41
  haplo.score.slide, 42
  plot.haplo.score.slide, 51
  print.haplo.score.merge, 58
  score.sim.control, 61
∗ **utilities**
  geno1to2, 8

anova.haplo.glm, 3
anova.haplo.glmlist *(*anova.haplo.glm*)*, 3

chisq.power, 4
chisq.sample.size *(*chisq.power*)*, 4

dglm.fit, 5

f.power, 5
f.sample.size *(*f.power*)*, 5
find.beta.qt.phase.known
        *(*find.haplo.beta.qt*)*, 6
find.haplo.beta.qt, 6, *34*
find.intercept.logistic
        *(*haplo.power.cc*)*, 30
find.intercept.qt.phase.known
        *(*find.haplo.beta.qt*)*, 6
fitted.haplo.glm, 7, *61*

geno.count.pairs, 7, *69*
geno1to2, 8
get.hapPair, 9
Ginv, 11
glm.control, *26*

haplo.binomial *(*dglm.fit*)*, 5
haplo.cc, 12, *55*
haplo.chistat *(*dglm.fit*)*, 5
haplo.design, 14
haplo.em, *8*, *13*, *14*, 15, *19*, *24*, *31*, *34*, *37*, *40*, *55*, *64*, *67–69*
haplo.em.control, *17*, 18, *26*, *37*, *40*
haplo.em.fitter, 19
haplo.enum *(*dglm.fit*)*, 5
haplo.glm, *3*, *7*, *13*, 20, *26*, *61*, *67*, *69*, *70*
haplo.glm.control, *24*, 25
haplo.group, *13*, 27, *41*
haplo.hash, 28
haplo.model.frame, *24*, 29
haplo.power.cc, 30, *34*
haplo.power.qt, *31*, 32
haplo.scan, 35, *57*
haplo.score, *13*, *19*, 37, *41*, *44*, *48*, *62*

haplo.score.glm *(*dglm.fit*)*, 5
haplo.score.merge, *13*, 41
haplo.score.podds *(*dglm.fit*)*, 5
haplo.score.slide, 42, *52*
hapPower.demo, 45
hla.demo, 46

locator.haplo, 47
locus, 48, *67*, *71*
louis.info, 49

mf.gindx *(*dglm.fit*)*, 5

na.geno.keep, 50

plot.haplo.score, *40*, 50
plot.haplo.score.slide, *44*, 51
plot.seqhap, 53, *64*
print.haplo.cc, *13*, 54
print.haplo.em, 55
print.haplo.glm *(*haplo.glm*)*, 20
print.haplo.group, 56
print.haplo.scan, 57
print.haplo.score, *40*, 57
print.haplo.score.merge, 58
print.haplo.score.slide, 59
print.seqhap, *54*, *64*
print.seqhap *(*seqhap*)*, 62
print.summary.haplo.glm
*(*summary.haplo.glm*)*, 68
printBanner, 59

residScaledGlmFit *(*dglm.fit*)*, 5
residuals.glm, *61*
residuals.haplo.glm, 60

score.sim.control, *37*, *40*, *44*, 61, *64*
seqhap, *54*, 62
seqhap.dat, 65
seqhap.pos *(*seqhap.dat*)*, 65
setupGeno, *17*, 66
sr.class *(*dglm.fit*)*, 5
sr.class<- *(*dglm.fit*)*, 5
summary.haplo.em, 67
summary.haplo.glm, 68
summaryGeno, *8*, 69

varfunc.glm.fit *(*dglm.fit*)*, 5
vcov.haplo.glm, 70

x.sexcheck, 70