Package 'SNPassoc'

April 13, 2026

Version 2.3.1

Date 2026-04-13

Depends R (>= 4.0.0)

Imports mvtnorm, parallel, survival, tidyr, plyr, ggplot2, poisbinom, rms, methods

Suggests testthat, knitr, rmarkdown, biomaRt, VariantAnnotation, GenomicRanges, IRanges, S4Vectors, org.Hs.eg.db, TxDb.Hsapiens.UCSC.hg19.knownGene

Title SNPs-Based Whole Genome Association Studies

Description Functions to perform most of the common analysis in genome association studies are implemented. These analyses include descriptive statistics and exploratory analysis of missing values, calculation of Hardy-Weinberg equilibrium, analysis of association based on generalized linear models (either for quantitative or binary traits), and analysis of multiple SNPs (haplotype and epistasis analysis). Permutation test and related tests (sum statistic and truncated product) are also implemented. Max-statistic and genetic risk-allele score exact distributions are also possible to be estimated. The methods are described in Gonzalez JR et al., 2007 <doi:10.1093/bioinformatics/btm025>. This version includes internal copies of functions from the archived 'haplo.stats' package to maintain functionality.

URL https://github.com/isglobal-brge/SNPassoc

License GPL (>= 2)

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.2.2

NeedsCompilation yes

Author Victor Moreno [aut], Juan R Gonzalez [aut] (ORCID: <https://orcid.org/0000-0003-3267-2146>), Dolors Pelegri [aut, cre] (ORCID: <https://orcid.org/0000-0002-5993-3003>)

Maintainer Dolors Pelegri <dolors.pelegri@isglobal.org>

Repository CRAN

Date/Publication 2026-04-13 10:10:09 UTC

Contents

`association` *Association analysis between a single SNP and a given phenotype*

Description

This function carries out an association analysis between a single SNP and a dependent variable (phenotype) under five different genetic models (inheritance patterns): codominant, dominant, recessive, overdominant and log-additive. The phenotype may be quantitative or categorical. In the second case (e.g. case-control studies) this variable must be of class 'factor' with two levels.

Usage

```
association(formula, data, model=c("all"), model.interaction=
       c("codominant"), subset, name.snp = NULL, quantitative =
       is.quantitative(formula,data), genotypingRate= 0,
       level = 0.95, ...)
```

Arguments

`formula`	a symbolic description of the model to be fited (a formula object). It might have either a continuous variable (quantitative traits) or a factor variable (case-control studies) as the response on the left of the ~ operator and a term corresponding to the SNP on the right. This term must be of class `snp` (e.g. ~snp(var), where var is a given SNP), and it is required. Terms with additional covariates on the right of the ~ operator may be added to fit an adjusted model (e.g., ~var1+var2+...+varN+SNP). The formula allows to incorporate more than one object of class `snp`. In that case, the analysis is done for the first SNP which appears in the formula adjusted by the others covariates and other additional SNPs.
`data`	a required dataframe of class 'setupSNP' containing the variables in the model.
`model`	a character string specifying the type of genetic model (mode of inheritance) for the SNP. This indicates how the genotypes should be collapsed. Possible values are "codominant", "dominant", "recessive", "overdominant", "additive" or "all". The default is "all" that fits the 5 possible genetic models. Only the first words are required, e.g "co", "do", etc.

`model.interaction`
: a character string specifying the type of genetic model (mode of inheritance) assumed for the SNP when it is included in a interaction term. Possible values are "codominant", "dominant", "recessive", "overdominant". The default is "codominant".

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process

`name.snp`
: optional label of the SNP variable to be printed.

`quantitative`
: logical value indicating whether the phenotype (that which is in the left of the operator ~ in 'formula' argument) is quantitative. The function 'is.quantitative' returns FALSE when the phenotype is a variable with two categories (i.e. indicating case-control status). Thus, it is not a required argument but it may be modified by the user.

`genotypingRate`
: minimum percentage of genotype rate for the SNP to be analyzed. Default is 0% (e.g. all SNPs are analyzed). This parameter should not be changed. It is used in the function 'WGassociation'.

`level`
: signification level for confidence intervals.

`...`
: Other arguments to be passed through glm function

Details

This function should be called by the user when we are interested in analyzing an unique SNP. It is recommended to use `WGassociation` function when more than one SNP is studied.

Value

For each genetic model (codominant, dominant, recessive, overdominant, and log-additive) the function gives a matrix with sample size and percentages for each genotype, the Odds Ratio and its 95% confidence interval (taking the most frequent homozygous genotype as the reference), the p-value corresponding to the likelihood ratio test obtained from a comparison with the null model, and the Akaike Information Criterion (AIC) of each genetic model. In the case of analyzing a quantitative trait, the function returns a matrix with sample size, mean and standard errors for each genotype, mean difference and its 95% confidence interval with respect to the most frequent homozygous genotype, the p-value obtained from an overall gene effect and the Akaike Information Criterion (AIC) of each genetic model.

When an interaction term (a categorical covariate with an SNP) is included in the model, three different tables are given. The first one correponds to the full interaction matrix where the ORs (or mean differences if a quantitative trait is analyzed) are expressed with respect to the non variant genotype and the first category of the covariate. The other two tables show the ORs and their 95% confidence intervals for both marginal models. P values for interaction and trend are also showed in the output.

References

JR Gonzalez, L Armengol, X Sole, E Guino, JM Mercader, X Estivill, V Moreno. SNPassoc: an R package to perform whole genome association studies. Bioinformatics, 2007;23(5):654-5.

Iniesta R, Guino E, Moreno V. Statistical analysis of genetic polymorphisms in epidemiological studies. Gac Sanit. 2005;19(4):333-41.

Elston RC. Introduction and overview. Statistical methods in genetic epidemiology. Stat Methods Med Res. 2000;9:527-41.

See Also

WGassociation

Examples

```
data(SNPs)

# first, we create an object of class 'setupSNP'
datSNP<-setupSNP(SNPs,6:40,sep="")

# case-control study, crude analysis
association(casco~snp10001, data=datSNP)

# case-control study, adjusted by sex and arterial blood pressure
association(casco~sex+snp10001+blood.pre, data=datSNP)


# quantitative trait, crude analysis
association(log(protein)~snp10001,data=datSNP)
# quantitative trait, adjusted by sex
association(log(protein)~snp10001+sex,data=datSNP)


#
# Interaction analysis
#

# Interaction SNP and factor
association(log(protein)~snp10001*sex+blood.pre, data=datSNP,
          model="codominant")

# Interaction SNP and SNP (codominant and codominant)
association(log(protein)~snp10001*factor(snp10002)+blood.pre,
          data=datSNP, model="codominant")

# Interaction SNP and SNP (dominant and recessive)
association(log(protein)~snp10001*factor(recessive(snp100019))+blood.pre,
          data=datSNP, model="dominant")
```

asthma *SNP data on asthma case-control study*

Description

data.frame with 51 SNPs and 6 epidemiological variables: country, gender, age, bmi, smoke ans case/control status.

Usage

```
data("asthma")
```

Format

The asthma data frame has 1578 rows (individuals) and 57 columns (variables) of data from a genetic study on asthma.

Value

An data.frame object.

Examples

```
data(asthma)
dim(asthma)
str(asthma)
```

Bonferroni.sig *Bonferroni correction of p values*

Description

This function shows the SNPs that are statistically significant after correcting for the number of tests performed (Bonferroni correction) for an object of class "WGassociation"

Usage

```
Bonferroni.sig(x, model = "codominant", alpha = 0.05,
     include.all.SNPs=FALSE)
```

Arguments

x	an object of class 'WGassociation'.
model	a character string specifying the type of genetic model (mode of inheritance). This indicantes how the genotypes should be collapsed when 'plot.summary' is TRUE. Possible values are "codominant", "dominant", "recessive", "overdominant", or "log-additive". The default is "codominant". Only the first words are required, e.g "co", "do", ...
alpha	nominal level of significance. Default is 0.05
include.all.SNPs	logical value indicating whether all SNPs are considered in the Bonferroni correction. That is, the number of performed tests is equal to the number of SNPs or equal to the number of SNPs where a p value may be computed. The default value is FALSE indicating that the number of tests is equal to the number of SNPs that are non Monomorphic and the rate of genotyping is greater than the percentage indicated in the GeneticModel.pval function.

Details

After deciding the genetic model, the function shows the SNPs that are statistically significant at alpha level corrected by the number of performed tests.

Value

A data frame with the SNPs and the p values for those SNPs that are statistically significant after Bonferroni correction

See Also

WGassociation

Examples

```
data(SNPs)
datSNP<-setupSNP(SNPs,6:40,sep="")
ans<-WGassociation(protein~1,data=datSNP,model="all")
Bonferroni.sig(ans, model="codominant", alpha=0.05, include.all.SNPs=FALSE)
```

GenomicControl *Population substructure*

Description

This function estimates an inflation (or deflation) factor, lambda, as indicated in the paper by Devlin et al. (2001) and corrects the p-values using this factor.

Usage

```
GenomicControl(x, snp.sel)
```

Arguments

x	an object of class 'WGassociation'.
snp.sel	SNPs used to compute lambda. Not required.

Details

This method is only valid for 2x2 tables. This means that the object of class 'WGassociation' might not have fitted the codominant model.

See reference for further details.

Value

The same object of class 'WGassociation' where the p-values have been corrected for genomic control.

References

B Devlin, K Roeder, and S.A. Bacanu. Unbiased Methods for Population Based Association Studies. Genetic Epidemiology (2001) 21:273-84

See Also

qqpval, WGassociation

Examples

```
data(SNPs)
datSNP<-setupSNP(SNPs,6:40,sep="")
res<-WGassociation(casco,datSNP,model=c("do","re","log-add"))

# Genomic Control
resCorrected<-GenomicControl(res)
```

---

getGeneSymbol *Get gene symbol from a list of SNPs*

Description

Get gene symbol from a list of SNPs

Usage

```
getGeneSymbol(
  x,
  snpCol = 1,
  chrCol = 2,
  posCol = 3,
  db = TxDb.Hsapiens.UCSC.hg19.knownGene
)
```

Arguments

x	data.frame containing: SNP name, chromosome and genomic position.
snpCol	column of x having the SNP name. Default is 1.
chrCol	column of x having the SNP chromosome. Default is 2.
posCol	column of x having the SNP position. Default is 3.
db	reference genome. Default is 'TxDb.Hsapiens.UCSC.hg19.knownGene'

Value

a data.frame having initial information and gene symbol

Examples

```
snps = c('rs58108140','rs189107123','rs180734498','rs144762171')
chr = c('chr1','chr1','chr1','chr1')
pos = c(10583, 10611, 13302, 13327)

x <- data.frame(snps, chr, pos )
if (requireNamespace("VariantAnnotation", quietly = TRUE)) {
    getGeneSymbol(x)
}
```

---

getNiceTable *Get Latex output*

Description

Create Latex output from association analyses

Usage

```
getNiceTable( x )
```

Arguments

x	WGassociation object.

Value

The R output of specific association analyses exported into LaTeX

`getSignificantSNPs` *Extract significant SNPs from an object of class 'WGassociation'*

Description

Extract significant SNPs from an object of class 'WGassociation' when genomic information is available

Usage

```
getSignificantSNPs(x, chromosome, model, sig = 1e-15)
```

Arguments

`x`	an object of class 'WGassociation'
`chromosome`	chromosome from which SNPs are extracted
`model`	genetic model from which SNPs are extracted
`sig`	statistical significance level. The default is 1e-15

Value

A list with the following components:

`names`	the name of SNPs
`column`	the columns corresponding to the SNPs in the original data frame

...

See Also

`WGassociation`

Examples

```
data(resHapMap)
# resHapMap contains the results for a log-additive genetic model

# to get the significant SNPs for chromosome 12
getSignificantSNPs(resHapMap,chromosome=12)
# to get the significant SNPs for chromosome 5
getSignificantSNPs(resHapMap,5)
# to get the significant SNPs for chromosome X at level 1e-8
getSignificantSNPs(resHapMap,5,sig=1e-8)
```

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-06)
```

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. The rank of the matrix is determined by the number of singular values that are at least as large as `max(svd)*eps`.

Value

A list with the following components:

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

References

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

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

Note

This function is a modified version of the original from the `haplo.stats` package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in `SNPassoc` to ensure continued functionality following the archival of `haplo.stats` from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for `SNPassoc` by Dolors Pelegrí-Sisó.

See Also

`svd`

Examples

```
x <- matrix(c(1, 2, 1, 2, 3, 2), ncol = 3)
save <- Ginv(x)
ginv.x <- save$Ginv
rank.x <- save$rank
```

`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. This function is a modified version of the original from the haplo.stats package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in SNPassoc to ensure continued functionality following the archival of haplo.stats from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for SNPassoc by Dolors Pelegrí-Sisó.

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(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.

Note

This function is a modified version of the original from the `haplo.stats` package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in `SNPassoc` to ensure continued functionality following the archival of `haplo.stats` from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for `SNPassoc` by Dolors Pelegrí-Sisó.

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.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.

Note

This function is a modified version of the original from the `haplo.stats` package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in `SNPassoc` to ensure continued functionality following the archival of `haplo.stats` from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for `SNPassoc` by Dolors Pelegrí-Sisó.

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

Note

This function is a modified version of the original from the `haplo.stats` package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in `SNPassoc` to ensure continued functionality following the archival of `haplo.stats` from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for `SNPassoc` by Dolors Pelegrí-Sisó.

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.interaction` *Haplotype interaction with a covariate*

Description

This function computes the ORs (or mean differences if a quantitative trait is analyzed) and their 95% confidence intervals corresponding to an interaction between the haplotypes and a categorical covariate

Usage

```
haplo.interaction(formula, data, SNPs.sel, quantitative =
  is.quantitative(formula, data), haplo.freq.min = 0.05, ...)
```

Arguments

formula	a symbolic description of the model to be fitted (a formula object). It might have either a continuous variable (quantitative traits) or a factor variable (case-control studies) as the response on the left of the ~ operator and a term corresponding to the interaction variable on the right indicated using 'interaction' function (e.g. ~int(var), where var is a factor variable) and it is required. Terms with additional covariates on the the right of the ~ operator may be added to fit an adjusted model (e.g., ~var1+var2+...+varN+int(var)).
data	an object of class 'setupSNP' containing the variables in the model and the SNPs that will be used to estimate the haplotypes.
SNPs.sel	a vector indicating the names of SNPs that are used to estimate the haplotypes
quantitative	logical value indicating whether the phenotype (which is on the left of the operator ~ in 'formula' argument) is quantitative. The function 'is.quantitative' returns FALSE when the phenotype is a variable with two categories (i.e. indicating case-control status). Thus, it is not a required argument but it may be modified by the user.
haplo.freq.min	control parameter for haplo.glm included in 'haplo.glm.control'. This parameter corresponds to 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 independently of any trait.
...	additional parameters for 'haplo.glm.control'.

Details

The function estimates the haplotypes for the SNPs indicated in the 'SNPs.sel' argument. Then, usign 'haplo.glm' function (from 'haplo.stats' library) estimates the interaction between these haplotypes and the covariate indicated in the formula by means of 'interaction' function.

Value

Three different tables are given. The first one corresponds to the full interaction matrix where the ORs (or mean differences if a quantitative trait is analyzed) are expressed with respect to the most frequent haplotype and the first category of the covariate. The other two tables show the ORs (or mean differences if a quantitative trait is analyzed) and their 95% confidence intervals for both marginal models. P values for interaction are also showed in the output.

Examples

```
# not Run
library(SNPassoc)

data(SNPs)
datSNP<-setupSNP(SNPs,6:40,sep="")
res <- haplo.interaction(log(protein)~int(sex), data=datSNP,
                   SNPs.sel=c("snp100019","snp10001","snp100029"))
res
```

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

Note

This function is a modified version of the original from the haplo.stats package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in SNPassoc to ensure continued functionality following the archival of haplo.stats from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for SNPassoc by Dolors Pelegrí-Sisó.

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.

Note

This function is a modified version of the original from the `haplo.stats` package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in `SNPassoc` to ensure continued functionality following the archival of `haplo.stats` from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for `SNPassoc` by Dolors Pelegrí-Sisó.

See Also

`haplo.em`, `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.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.

Note

This function is a modified version of the original from the `haplo.stats` package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in `SNPassoc` to ensure continued functionality following the archival of `haplo.stats` from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for `SNPassoc` by Dolors Pelegrí-Sisó.

See Also

`haplo.score`, `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")
```

HapMap *SNPs from HapMap project*

Description

Information about 9307 SNPs from the HapMap project belonging to 22 chromosomes. Information about two different population is available: European population (CEU) and Yoruba (YRI). The genomic information (names of SNPs, chromosomes and genetic position) is also available in a data frame called 'HapMap.SNPs.pos'.

Usage

```
data(HapMap)
```

Format

A data frame with 120 observations on the 9808 variables (SNPs) and one variable called 'group' indicating the population.

Source

HapMap project (http://www.hapmap.org)

Examples

```
data(HapMap)
```

`HapMap.SNPs.pos` *SNPs from HapMap project*

Description

Information about 9307 SNPs from the HapMap project belonging to 22 chromosomes. Information about two different population is available: European population (CEU) and Yoruba (YRI). The genomic information (names of SNPs, chromosomes and genetic position) is also available in a data frame called 'HapMap.SNPs.pos'.

Usage

```
data(HapMap.SNPs.pos)
```

Format

A data frame with 120 observations on the 9808 variables (SNPs) and one variable called 'group' indicating the population.

Source

HapMap project (http://www.hapmap.org)

Examples

```
data(HapMap.SNPs.pos)
```

`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)
```

`inheritance` *Collapsing (or recoding) genotypes into different categories (generally two) depending on a given genetic mode of inheritance*

Description

codominant function recodifies a variable having genotypes depending on the allelic frequency in descending order.

dominant, recessive, and overdominant functions collapse the three categories of a given SNP into two categories as follows: Let 'AA', 'Aa', and 'aa' be the three genotypes. After determining the most frequent allele (let's suppose that 'A' is the major allele) the functions return a vector with to categories as follows. dominant: 'AA' and 'Aa-aa'; recessive: 'AA-Aa' and 'aa'; overdominant: 'AA-aa' vs 'Aa'.

additive function creates a numerical variable, 1, 2, 3 corresponding to the three genotypes sorted out by descending allelic frequency (this model is referred as log-additive).

Usage

```
codominant(o)
dominant(o)
recessive(o)
overdominant(o)
additive(o)
```

Arguments

`o`	categorical covariate having genotypes

Value

A snp object collapsing genotypes into different categories depending on a given genetic mode of inheritance

Examples

```
data(SNPs)
dominant(snp(SNPs$snp10001,sep=""))
overdominant(snp(SNPs$snp10001,sep=""))
```

int *Identify interaction term*

Description

This is a special function used for 'haplo.interaction' function. It identifies the variable that will interact with the haplotype estimates. Using `int()` in a formula implies that the interaction term between this variable and haplotypes is included in 'haplo.glm' function.

Usage

```
int(x)
```

Arguments

x	A factor variable.

Value

x

See Also

haplo.interaction

Examples

```
library(SNPassoc)

data(SNPs)
datSNP<-setupSNP(SNPs, 6:40, sep = "")
mod <- haplo.interaction(casco~int(sex)+blood.pre, data = datSNP,
                         SNPs.sel = c("snp10001","snp10004","snp10005"))
```

interactionPval *Two-dimensional SNP analysis for association studies*

Description

Perform a two-dimensional SNP analysis (interaction) for association studies with possible allowance for covariate

Usage

```
interactionPval(formula, data, quantitative =
          is.quantitative(formula, data), model = "codominant")
```

Arguments

formula	a formula object. It might have either a continuous variable (quantitative traits) or a factor variable (case-control study) as the response on the left of the ~ operator and the terms corresponding to the covariates to be adjusted. A crude analysis is performed indicating ~1
data	a required object of class 'setupSNP'.
quantitative	logical value indicating whether the phenotype (those which is in the left of the operator ~ in 'formula' argument) is quantitative. The function 'is.quantitative' returns FALSE when the phenotype is a variable with two categories (i.e. indicating case-control status). Thus, it is not a required argument but it may be modified by the user.
model	a character string specifying the type of genetic model (mode of inheritance). This indicates how the genotypes should be collapsed. Possible value are "codominant", "dominant", "recessive", "overdominant" or "log-additive". The default is "codominant". Only the first words are required, e.g "co", "do", "re", "ov", "log"

Details

The 'interactionPval' function calculates, for each pair of SNPs (i,j), the likelihood underling the null model L0, the likelihood under each of the single-SNP, L(i) and L(j), the likelihood under an additive SNP model La(i,j), and the likelihood under a full SNP model (including SNP-SNP interaction), Lf(i,j).

The upper triangle in matrix from this function contains the p values for the interaction (epistasis) log-likelihood ratio test, LRT, LRTij = -2 (log Lf(i,j) - log La(i,j))

The diagonal contains the p values from LRT for the crude effect of each SNP, LRTii = -2 (log L(i) - log L0)

The lower triangle contains the p values from LRT comparing the two-SNP additive likelihood to the best of the single-SNP models, LRTji = -2 (log La(i,j) - log max(L(i),L(j)))

In all cases the models including the SNPs are adjusted by the covariates indicated in the 'formula' argument. This method is used either for quantitative traits and dicotomous variables (case-control studies).

Value

The 'interactionPval' function returns a matrix of class 'SNPinteraction' containing the p values corresponding to the different likelihood ratio tests above describe.

Methods defined for 'SNPinteraction' objects are provided for print and plot. The plot method uses 'image' to plot a grid of p values. The upper triangle contains the interaction (epistasis) p values from LRT. The content in the lower triangle is the p values from the LRT comparing the additive model with the best single model. The diagonal contains the main effects pvalues from LRT. The 'plot.SNPinteraction' function also allows the user to plot the SNPs sorted by genomic position and with the information about chromosomes as in the 'plotMissing' function.

Note

two-dimensional SNP analysis on a dense grid can take a great deal of computer time and memory.

References

JR Gonzalez, L Armengol, X Sole, E Guino, JM Mercader, X Estivill, V Moreno. SNPassoc: an R package to perform whole genome association studies. Bioinformatics, 2007;23(5):654-5.

See Also

setupSNP

Examples

```
data(SNPs)
datSNP <- setupSNP(SNPs, 6:40, sep="")[1:10,]

ansCod <- interactionPval(log(protein)~sex,datSNP)
print(ansCod)
plot(ansCod)
```

`intervals` *Print ORs and 95% confidence intervals for an object of class 'haplo.glm'*

Description

Print ORs and confidence intervals for an object of class 'haplo.glm'

Usage

```
intervals(o, level=.95, ...)
```

Arguments

`o`	object of class 'haplo.glm'
`level`	significance level. Default is 95 percent
`...`	other arguments

Value

intervals object with ORs and 95% confidence intervals for an object of class 'haplo.glm'

Examples

```
library(SNPassoc)

data(asthma, package = "SNPassoc")

asthma.s <- setupSNP(data=asthma, colSNPs=7:ncol(asthma), sep="")
trait <- asthma.s$casecontrol
snpsH <- c("rs714588", "rs1023555",  "rs898070")
genoH <- make.geno(asthma.s, snpsH)

mod <- haplo.glm( trait ~ genoH,
                  family="binomial",
                  locus.label=snpsH,
                  allele.lev=attributes(genoH)$unique.alleles,
                  control = haplo.glm.control(haplo.freq.min=0.05))
intervals(mod)
summary(mod)
```

LD *max-statistic for a 2x3 table*

Description

Compute pairwise linkage disequilibrium between genetic markers

Usage

```
LD(g1, ...)

## S3 method for class 'snp'
LD(g1, g2, ...)

## S3 method for class 'setupSNP'
LD(g1, SNPs, ...)


LDplot(x, digits = 3, marker, distance, which = c("D", "D'",
    "r", "X^2", "P-value", "n", " "), ...)

LDtable(x, colorcut = c(0, 0.01, 0.025, 0.05, 0.1, 1),
       colors = heat.colors(length(colorcut)),
       textcol = "black", digits = 3, show.all = FALSE,
       which = c("D", "D'", "r", "X^2", "P-value", "n"),
       colorize = "P-value", cex, ...)
```

Arguments

g1	genotype object or dataframe containing genotype objects
g2	genotype object (ignored if g1 is a dataframe)
SNPs	columns containing SNPs
x	LD or LD.data.frame object
digits	Number of significant digits to display
which	Name(s) of LD information items to be displayed
colorcut	P-value cutoffs points for colorizing LDtable
colors	Colors for each P-value cutoff given in 'colorcut' for LDtable
textcol	Color for text labels for LDtable
marker	Marker used as 'comparator' on LDplot. If omitted separate lines for each marker will be displayed
distance	Marker location, used for locating of markers on LDplot.
show.all	If TRUE, show all rows/columns of matrix. Otherwise omit completely blank rows/columns.
colorize	LD parameter used for determining table cell colors
cex	Scaling factor for table text. If absent, text will be scaled to fit within the table cells.
...	Optional arguments ('plot.LD.data.frame' passes these to 'LDtable' and 'LDplot').

Value

None

Author(s)

functions adapted from LD, LDtable and LDplot in package genetics by Gregory Warnes et al. (warnes@bst.rochester.edu)

References

genetics R package by Gregory Warnes et al. (warnes@bst.rochester.edu)

See Also

setupSNP snp

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

Note

This function is a modified version of the original from the haplo.stats package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in SNPassoc to ensure continued functionality following the archival of haplo.stats from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for SNPassoc by Dolors Pelegrí-Sisó.

make.geno *Create a group of locus objects from some SNPs, assign to 'model.matrix' class.*

Description

This function prepares the CRITICAL element corresponding to matrix of genotypes necessary to be included in 'haplo.glm' function.

Usage

```
make.geno(data, SNPs.sel)
```

Arguments

data	an object of class 'setupSNP' containing the the SNPs that will be used to estimate the haplotypes.
SNPs.sel	a vector indicating the names of SNPs that are used to estimate the haplotypes

Value

the same as 'setupGeno' function, from 'haplo.stats' library, returns

See Also

snp

Examples

```
## Not run:
data(SNPs)
# first, we create an object of class 'setupSNP'
datSNP<-setupSNP(SNPs,6:40,sep="")
geno<-make.geno(datSNP,c("snp10001","snp10002","snp10003"))
## End(Not run)
```

---

maxstat *max-statistic for a 2x3 table*

Description

Computes the asymptotic p-value for max-statistic for a 2x3 table

Usage

```
maxstat(x, ...)

## Default S3 method:
maxstat(x, y, ...)

## S3 method for class 'table'
maxstat(x, ...)

## S3 method for class 'setupSNP'
maxstat(x, y, colSNPs=attr(x,"colSNPs"), ...)

## S3 method for class 'matrix'
maxstat(x, ...)
```

Arguments

x	a numeric matrix with 2 rows (cases/controls) and 3 colums (genotypes) or a vector with case/control status or an object of class 'setupSNP'.

y	an optional numeric vector containing the information for a given SNP. In this case 'x' argument must contain a vector indicarting case/control status. If 'x' argument is an object of class 'setupSNP' this argument migth be the name of the variable containing case/control information.
colSNPs	a vector indicating which columns contain those SNPs to compute max-statistic. By default max-statistic is computed for those SNPs specified when the object of class 'setupSNP' was created.
...	further arguments to be passed to or from methods.

Value

A matrix with the chi-square statistic for dominant, recessive, log-additive and max-statistic and its asymptotic p-value.

References

Gonzalez JR, Carrasco JL, Dudbridge F, Armengol L, Estivill X, Moreno V. Maximizing association statistics over genetic models (2007). Submitted

Sladek R, Rocheleau G, Rung J et al. A genome-wide association study identifies novel risk loci for type 2 diabetes (2007). Nature 445, 881-885

See Also

setupSNP

Examples

```
# example from Sladek et al. (2007) for the SNP rs1111875
 tt<-matrix(c(77,298,310,122,316,231),nrow=2,ncol=3,byrow=TRUE)
 maxstat(tt)

 data(SNPs)
 maxstat(SNPs$casco,SNPs$snp10001)
 myDat<-setupSNP(SNPs,6:40,sep="")
 maxstat(myDat,casco)
```

---

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

Note

This function is a modified version of the original from the haplo.stats package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in SNPassoc to ensure continued functionality following the archival of haplo.stats from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for SNPassoc by Dolors Pelegrí-Sisó.

---

odds *Extract odds ratios, 95% CI and pvalues*

Description

Extract odds ratios, 95

Usage

```
odds(x, model=c("log-additive", "dominant", "recessive", "overdominant", "codominant"),
     sorted=c("no","p-value","or"))
```

Arguments

x	an object of class 'WGassociation' output of WGassociation
model	model to be extracted. Only first one is used. The first letter is enough, low or upper case.
sorted	Sort the output by P value or OR.

Value

A matrix with OR 95% CI (lower, upper) and P value for the selected model. For codominant model, the OR and 95%CI are given for heterozygous and homozigous.

References

JR Gonzalez, L Armengol, X Sole, E Guino, JM Mercader, X Estivill, V Moreno. SNPassoc: an R package to perform whole genome association studies. Bioinformatics, 2007;23(5):654-5.

Examples

```
data(SNPs)
datSNP<-setupSNP(SNPs,6:40,sep="")
ans<-WGassociation(casco~1,data=datSNP,model="all")
odds(ans)
```

`permTest` *Permutation test analysis*

Description

This function extract the p values for permutation approach performed using scanWGassociation function

Usage

```
permTest(x, method="minimum", K)
```

Arguments

`x`	a required object of class 'WGassociation' with the attribute 'permTest'. See details
`method`	statistic used in the permutation test. The default is 'minimum' but 'rtp' (rank truncated product) is also available.
`K`	number of the K most significant p values from the total number of test performed (e.g number of SNPs) used to compute the rank truncated product. This argument is only required when method='rtp'. See references

Details

This function extract the p values from an object of class 'WGassociation'. This object migth be obtained using the funcion called 'scanWGassociation' indicating the number of permutations in the argument 'nperm'.

Value

An object of class 'permTest'.

'print' returns a summary indicating the number of SNPs analyzed, the number of valid SNPs (those non-Monomorphic and that pass the calling rate), the p value after Bonferroni correction, and the p values based on permutation approach. One of them is based on considering the empirical percentil for the minimum p values, and the another one on assuming that the minimum p values follow a beta distribution.

'plot' produces a plot of the empirical distribution for the minimum p values (histogram) and the expected distribution assuming a beta distribution. The corrected p value is also showed in the plot.

See examples for further illustration about all previous issues.

References

Dudbridge F, Gusnanto A and Koeleman BPC. Detecting multiple associations in genome-wide studies. Human Genomics, 2006;2:310-317.

Dudbridge F and Koeleman BPC. Efficient computation of significance levels for multiple associations in large studies of correlated data, including genomewide association studies. Am J Hum Genet, 2004;75:424-435.

JR Gonzalez, L Armengol, X Sole, E Guino, JM Mercader, X Estivill, V Moreno. SNPassoc: an R package to perform whole genome association studies. Bioinformatics, 2007;23(5):654-5.

See Also

scanWGassociation

Examples

```
library(SNPassoc)

data(asthma, package = "SNPassoc")
asthma.s <- setupSNP(data=asthma, colSNPs=7:ncol(asthma), sep="")

ans <- WGassociation(casecontrol, data=asthma.s)
```

---

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.

Note

This function is a modified version of the original from the `haplo.stats` package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in `SNPassoc` to ensure continued functionality following the archival of `haplo.stats` from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for `SNPassoc` by Dolors Pelegrí-Sisó.

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")

  ## try: locator.haplo(1)
```

`plot.WGassociation` *Function to plot -log p values from an object of class 'WGassociation'*

Description

Function to plot -log p values from an object of class 'WGassociation'

Usage

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

Arguments

`x`	an object of class 'WGassociation'
`...`	other graphical parameters

Details

A panel with different plots (one for each mode of inheritance) are plotted. Each of them represents the -log(p value) for each SNP. Two horizontal lines are also plotted. One one them indicates the nominal statistical significance level whereas the other one indicates the statistical significance level after Bonferroni correction.

Value

No return value, just the plot

References

JR Gonzalez, L Armengol, X Sole, E Guino, JM Mercader, X Estivill, V Moreno. SNPassoc: an R package to perform whole genome association studies. Bioinformatics, 2007;23(5):654-5.

See Also

`association` `setupSNP` `WGassociation`

Examples

```
library(SNPassoc)

data(asthma, package = "SNPassoc")
asthma.s <- setupSNP(data=asthma, colSNPs=7:ncol(asthma), sep="")

ans <- WGassociation(casecontrol, data=asthma.s)

plot(ans)
```

plotMissing *Plot of missing genotypes*

Description

Plot a grid showing which genotypes are missing

Usage

```
plotMissing(x, print.labels.SNPs = TRUE,
        main = "Genotype missing data", ...)
```

Arguments

x	an object of class 'setupSNP'
print.labels.SNPs	should labels of SNPs be printed?
main	title to place on plot
...	extra arguments of 'image' function

Details

This function uses 'image' function to plot a grid with black pixels where the genotypes are missing.

Value

No return value, just the plot

See Also

setupSNP

Examples

```
data(SNPs)
data(SNPs.info.pos)
ans<-setupSNP(SNPs,colSNPs=6:40,sep="")
plotMissing(ans)

# The same plot with the SNPs sorted by genomic position and
# showing the information about chromosomes

ans<-setupSNP(SNPs,colSNPs=6:40,sort=TRUE,SNPs.info.pos,sep="")
plotMissing(ans)
```

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

Note

This function is a modified version of the original from the haplo.stats package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in SNPassoc to ensure continued functionality following the archival of haplo.stats from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for SNPassoc by Dolors Pelegrí-Sisó.

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
# ========================================
```

qqpval *Functions for inspecting population substructure*

Description

This function plots ranked observed p values against the corresponding expected p values in -log scale.

Usage

```
qqpval(p, pch=16, col=4, ...)
```

Arguments

p	a vector of p values
pch	symbol to use for points
col	color for points
...	other plot arguments

Value

No return value, just the plot

See Also

GenomicControl, WGassociation

Examples

```
data(SNPs)
datSNP<-setupSNP(SNPs,6:40,sep="")
res<-WGassociation(casco,datSNP,model=c("do","re","log-add"))

# observed vs expected p values for recessive model
qqpval(recessive(res))
```

related *Get related samples*

Description

Get related samples

Usage

```
related(x)
```

Arguments

x	An object obtained from SNPrelate package.

Value

A matrix with related individuals.

Examples

```
library(SNPassoc)
data(SNPs)
```

resHapMap *SNPs from HapMap project*

Description

Information about 9307 SNPs from the HapMap project belonging to 22 chromosomes. Information about two different population is available: European population (CEU) and Yoruba (YRI). The genomic information (names of SNPs, chromosomes and genetic position) is also available in a data frame called 'HapMap.SNPs.pos'.

Usage

```
data(resHapMap)
```

Format

A data frame with 120 observations on the 9808 variables (SNPs) and one variable called 'group' indicating the population.

Source

HapMap project (http://www.hapmap.org)

Examples

```
data(resHapMap)
```

`scanWGassociation` *Whole genome association analysis*

Description

This function is obsolete due to some problems with gfotran compiler. Use 'WGassociation' function instead or send an e-mail to the maintainer for receiving a version including this function

---

`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.

Note

This function is a modified version of the original from the `haplo.stats` package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in `SNPassoc` to ensure continued functionality following the archival of `haplo.stats` from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for `SNPassoc` by Dolors Pelegrí-Sisó.

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)
```

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. This function is a modified version of the original from the haplo.stats package (Copyright 2003 Mayo Foundation for Medical Education and Research). It has been included in SNPassoc to ensure continued functionality following the archival of haplo.stats from CRAN.

Author(s)

Original code by Daniel J. Schaid. Adapted for SNPassoc by Dolors Pelegrí-Sisó.

See Also

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
```

---

setupSNP *Convert columns in a dataframe to class 'snp'*

Description

setupSNP Convert columns in a dataframe to class 'snp'

summary.setupSNP gives a summary for an object of class 'setupSNP' including allele names, major allele frequencie, an exact thest of Hardy-Weinberg equilibrium and percentage of missing genotypes

Usage

```
setupSNP(data, colSNPs, sort = FALSE, info, sep = "/", ...)
```

Arguments

data	dataframe containing columns with the SNPs to be converted
colSNPs	Vector specifying which columns contain SNPs data
sort	should SNPs be sorted. Default is FALSE
info	if sort is TRUE a dataframe containing information about the SNPs regarding their genomic position and the gene where they are located
sep	character separator used to divide alleles in the genotypes
...	optional arguments

Value

a dataframe of class 'setupSNP' containing converted SNP variables. All other variables will be unchanged.

References

JR Gonzalez, L Armengol, X Sole, E Guino, JM Mercader, X Estivill, V Moreno. SNPassoc: an R package to perform whole genome association studies. Bioinformatics, 2007;23(5):654-5.

See Also

snp

Examples

```
 data(SNPs)
 myDat<-setupSNP(SNPs,6:40,sep="")


#sorted SNPs and having genomic information
 data(SNPs.info.pos)
 myDat.o<-setupSNP(SNPs,6:40,sep="",sort=TRUE, info=SNPs.info.pos)

# summary
 summary(myDat.o)

# plot one SNP
  plot(myDat,which=2)
```

---

snp *SNP object*

Description

snp creates an snp object

is returns TRUE if x is of class 'snp'

as attempts to coerce its argument into an object of class 'snp'

reorder change the reference genotype

summary gives a summary for an object of class 'snp' including genotype and allele frequencies and an exact thest of Hardy-Weinberg equilibrium

plot gives a summary for an object of class 'snp' including genotype and allele frequencies and an exact thest of Hardy-Weinberg equilibrium in a plot. Barplot or pie are allowed

[.snp is a copy of [.factor modified to preserve all attributes

Usage

```
snp(x, sep = "/", name.genotypes, reorder="common",
  remove.spaces = TRUE, allow.partial.missing = FALSE)

is.snp(x)

as.snp(x, ...)

## S3 method for class 'snp'
additive(o)
```

Arguments

x	either an object of class 'snp' or an object to be converted to class 'snp'
sep	character separator used to divide alleles when x is a vector of strings where each string holds both alleles. The default is "/". See below for details.
name.genotypes	the codes for the genotypes. This argument may be useful when genotypes are coded using three different codes (e.g., 0,1,2 or hom1, het, hom2)
reorder	how should genotypes within an individual be reordered. Possible values are 'common' or 'minor'. The default is reorder="common". In that case, alleles are sorted within each individual by common homozygous.
remove.spaces	logical indicating whether spaces and tabs will be removed from the genotypes before processing
allow.partial.missing	logical indicating whether one allele is permitted to be missing. When set to 'FALSE' both alleles are set to 'NA' when either is missing.
o	an object of class 'snp' to be coded as a linear covariate: 0,1,2
...	optional arguments

Details

SNP objects hold information on which gene or marker alleles were observed for different individuals. For each individual, two alleles are recorded.

The snp class considers the stored alleles to be unordered , i.e., "C/T" is equivalent to "T/C". It assumes that the order of the alleles is not important.

When snp is called, x is a character vector, and it is assumed that each element encodes both alleles. In this case, if sep is a character string, x is assumed to be coded as "Allele1<sep>Allele2". If sep is a numeric value, it is assumed that character locations 1:sep contain allele 1 and that remaining locations contain allele 2.

additive.snp recodes the SNPs for being analyzed as a linear covariate (codes 0,1,2)

Value

The snp class extends "factor" where the levels is a character vector of possible genotype values stored coded by paste(allele1, "", allele2, sep="/")

References

JR Gonzalez, L Armengol, X Sole, E Guino, JM Mercader, X Estivill, V Moreno. SNPassoc: an R package to perform whole genome association studies. Bioinformatics, 2007;23(5):654-5.

See Also

association

Examples

```
# some examples of snp data in different formats

dat1  <- c("21", "21", "11", "22", "21",
                   "22", "22", "11", "11", NA)
ans1  <- snp(dat1,sep="")
ans1

dat2 <- c("A/A","A/G","G/G","A/G","G/G",
                   "A/A","A/A","G/G",NA)
ans2  <- snp(dat2,sep="/")
ans2

dat3 <- c("C-C","C-T","C-C","T-T","C-C",
                   "C-C","C-C","C-C","T-T",NA)
ans3 <- snp(dat3,sep="-")
ans3


dat4 <- c("het","het","het","hom1","hom2",
                   "het","het","hom1","hom1",NA)
ans4 <- snp(dat4,name.genotypes=c("hom1","het","hom2"))
ans4


# summary
summary(ans3)

# plots

plot(ans3)
plot(ans3,type=pie)
plot(ans3,type=pie,label="SNP 10045")
```

SNPs *SNPs in a case-control study*

Description

SNPs data.frame contains selected SNPs and other clinical covariates for cases and controls in a case-control study

SNPs.info.pos data.frame contains the names of the SNPs included in the data set 'SNPs' including their chromosome and their genomic position

Usage

```
data(SNPs)
```

Format

'SNPs' data.frame contains the following columns:

id	identifier of each subject
casco	case or control status: 0-control, 1-case
sex	gender: Male and Female
blood.pre	arterial blood presure
protein	protein levels
snp10001	SNP 1
snp10002	SNP 2
...	...
snp100036	SNP 36

Source

Data obtained from our department. The reference and details will be supplied after being published.

SNPs.info.pos *SNPs in a case-control study*

Description

SNPs data.frame contains selected SNPs and other clinical covariates for cases and controls in a case-control study

SNPs.info.pos data.frame contains the names of the SNPs included in the data set 'SNPs' including their chromosome and their genomic position

Usage

```
data(SNPs.info.pos)
```

Format

'SNPs.info.pos' data.frame contains the following columns: A data frame with 35 observations on the following 3 variables.

`snp` name of SNP

`chr` name of chromosome

`pos` genomic position

Source

Data obtained from our department. The reference and details will be supplied after being published.

`sortSNPs` *Sort a vector of SNPs by genomic position*

Description

This function sorts a vector with the position of SNPs in a data frame using another data frame which contains information about SNPs, their chromosome, and their genomic position

Usage

```
sortSNPs(data, colSNPs, info)
```

Arguments

`data`	a required data frame with the SNPs
`colSNPs`	a required vector indicating which columns of 'data' contains genotype data
`info`	a required data frame with genomic information for the SNPs (chromosome and position). The first column must have the SNPs, the second one the chromosome and the third one the genomic position.

Details

First of all, the function obtains a vector with the SNPs sorted using the data frame with the genomic positions (see 'dataSNPs.pos' argument). Then, the columns which indicate where the information about the genotypes is in our data frame, are sorted using this vector.

This information is useful when `WGassociation` function is called since it allow the user to summaryze the results with the SNPs sorted by genomic position

Value

a vector indicating the colums where the SNPs are recorded in our data frame ('data' argument), sorted using the genomic positions listed in 'dataSNPs.pos' argument)

Examples

```
#
# data(SNPs)
# data(SNPs.info.pos)
# colSNPs.order<-sortSNPs(SNPs,c(6:40),SNPs.info.pos)
#
```

`Table.mean.se` *Descriptive sample size, mean, and standard error*

Description

This function computes sample size, mean and standard error of a quantitative trait for each genotype (or combination of genotypes)

Usage

```
Table.mean.se(var, dep, subset = !is.na(var))
```

Arguments

`var`	quantitative trait
`dep`	variable with genotypes or any combination of them
`subset`	an optional vector specifying a subset of observations to be used in the descriptive analysis

Value

`tp`	A matrix giving sample size (n), median (me) and standard error (se) for each genotype

See Also

`Table.N.Per`

Examples

```
data(SNPs)
# sample size, mean age and standard error for each genotype
Table.mean.se(SNPs$snp10001,SNPs$protein)

# The same table for a subset (males)
Table.mean.se(SNPs$snp10001,SNPs$protein,SNPs$sex=="Male")

# The same table assuming a dominant model
Table.mean.se(dominant(snp(SNPs$snp10001,sep="")),SNPs$protein,SNPs$sex=="Male")
```

Table.N.Per *Descriptive sample size and percentage*

Description

This function computes sample size and percentage for each category of a categorical trait (e.g. case-control status) for each genotype (or combination of genotypes).

Usage

```
Table.N.Per(var, dep, subset = !is.na(var))
```

Arguments

var	categorical trait.
dep	variable with genotypes or any combination of them
subset	an optional vector specifying a subset of observations to be used in the descriptive analysis.

Value

tp	A matrix giving sample size (n),and the percentage (%) for each level of the categorical trait for each genotype

See Also

Table.mean.se

Examples

```
data(SNPs)
#sample size and percentage of cases and controls for each genotype
Table.N.Per(SNPs$snp10001,SNPs$casco)

# The same table for a subset (males)
Table.N.Per(SNPs$snp10001,SNPs$casco,SNPs$sex=="Male")

# The same table assuming a dominant model
Table.N.Per(dominant(snp(SNPs$snp10001,sep="")),SNPs$casco,SNPs$sex=="Male")
```

tableHWE *Test for Hardy-Weinberg Equilibrium*

Description

Test the null hypothesis that Hardy-Weinberg equilibrium holds in cases, controls and both populations.

`print` print the information. Number of digits, and significance level can be changed

Usage

```
tableHWE(x, strata, ...)
```

Arguments

`x`	an object of class 'setupSNP'
`strata`	a factor variable for a stratified analysis
`...`	optional arguments

Details

This function calculates the HWE test for those variables of class 'snp' in the object x of class 'setupSNP'

Value

A matrix with p values for Hardy-Weinberg Equilibrium

Author(s)

This function is based on an R function which computes an exact SNP test of Hardy-Weinberg Equilibrium written by Wigginton JE, Cutler DJ and Abecasis GR available at http://csg.sph.umich.edu/abecasis/Exact/r_instruct.html

References

Wigginton JE, Cutler DJ and Abecasis GR (2005). A note on exact tests of Hardy-Weinberg equilibrium. Am J Hum Genet 76:887-93

See Also

setupSNP

Examples

```
data(SNPs)
ans<-setupSNP(SNPs,6:40,sep="")
res<-tableHWE(ans)
print(res)
#change the significance level showed in the flag column
print(res,sig=0.001)


#stratified analysis
res<-tableHWE(ans,ans$sex)
print(res)
```

`WGassociation` *Whole genome association analysis*

Description

This function carries out a whole genome association analysis between the SNPs and a dependent variable (phenotype) under five different genetic models (inheritance patterns): codominant, dominant, recessive, overdominant and log-additive. The phenotype may be quantitative or categorical. In the second case (e.g. case-control studies) this variable must be of class 'factor' with two levels.

Usage

```
WGassociation(formula, data, model = c("all"),
              quantitative = is.quantitative(formula, data),
              genotypingRate = 80, level = 0.95, ...)
```

Arguments

`formula`	either a symbolic description of the model to be fited (a formula object) without the SNP or the name of response variable in the case of fitting single models (e.g. unadjusted models). It might have either a continuous variable (quantitative traits) or a factor variable (case-control studies) as the response on the left of the ~ operator and terms with additional covariates on the right of the ~ operator may be added to fit an adjusted model (e.g., ~var1+var2+...+varN+SNP). See details
`data`	a required dataframe of class 'setupSNP' containing the variables in the model and the SNPs
`model`	a character string specifying the type of genetic model (mode of inheritance) for the SNP. This indicates how the genotypes should be collapsed. Possible values are "codominant", "dominant", "recessive", "overdominant", "log-additive" or "all". The default is "all" that fits the 5 possible genetic models. Only the first words are required, e.g "co", "do", etc.

quantitative	logical value indicating whether the phenotype (that which is in the left of the operator ~ in 'formula' argument) is quantitative. The function 'is.quantitative' returns FALSE when the phenotype is a variable with two categories (i.e. indicating case-control status). Thus, it is not a required argument but it may be modified by the user.
genotypingRate	minimum percentage of genotype rate for a given SNP to be included in the analysis. Default is 80%.
level	signification level for confidence intervals. Defaul 95%.
...	Other arguments to be passed through glm function

Details

This function assesses the association between the response variable included in the left side in the 'formula' and the SNPs included in the 'data' argument adjusted by those variables included in the right side of the 'formula'. Different genetic models may be analyzed using 'model' argument.

Value

An object of class 'WGassociation'.

'summary' returns a summary table by groups defined in info (genes/chromosomes).

'WGstats' returns a detailed output, similar to the produced by `association`.

'pvalues' and 'print' return a table of p-values for each genetic model for each SNP. The first column indicates whether a problem with genotyping is present.

'plot' produces a plot of p values in the -log scale. See `plot.WGassociation` for further details.

'labels' returns the names of the SNPs analyzed.

The functions 'codominat', 'dominant', 'recessive', 'overdominant' and 'additive' are used to obtain the p values under these genetic models.

See examples for further illustration about all previous issues.

References

JR Gonzalez, L Armengol, X Sole, E Guino, JM Mercader, X Estivill, V Moreno. SNPassoc: an R package to perform whole genome association studies. Bioinformatics, 2007;23(5):654-5.

See Also

`getSignificantSNPs` `association` `WGstats` `setupSNP` `plot.WGassociation`

Examples

```
data(SNPs)
datSNP<-setupSNP(SNPs,6:40,sep="")
ansAll<-WGassociation(protein~1,data=datSNP,model="all")

# In that case the formula is not required. You can also write:
# ansAll<-WGassociation(protein,data=datSNP,model="all")
```

```
#only codominant and log-additive
ansCoAd<-WGassociation(protein~1,data=datSNP,model=c("co","log-add"))

#for printing p values
print(ansAll)
print(ansCoAd)

#for obtaining a matrix with the p palues
pvalAll<-pvalues(ansAll)
pvalCoAd<-pvalues(ansCoAd)

# when all models are fitted and we are interested in obtaining
# p values for different genetic models

# codominant model
pvalCod<-codominant(ansAll)

# recessive model
pvalRec<-recessive(ansAll)

# and the same for additive, dominant or overdominant


#summary
summary(ansAll)

#for a detailed report
WGstats(ansAll)

#for plotting the p values
plot(ansAll)
```

Index