Package 'genetics'

October 13, 2022

Title Population Genetics

Version 1.3.8.1.3

Date 2012-11-26

Author Gregory Warnes, with contributions from Gregor Gorjanc, Friedrich Leisch, and Michael Man.

Maintainer Gregory Warnes <greg@warnes.net>

Depends combinat, gdata, gtools, MASS, mvtnorm

Description Classes and methods for handling genetic data. Includes classes to represent genotypes and haplotypes at single markers up to multiple markers on multiple chromosomes. Function include allele frequencies, flagging homo/heterozygotes, flagging carriers of certain alleles, estimating and testing for Hardy-Weinberg disequilibrium, estimating and testing for linkage disequilibrium, ...

biocViews Genetics

License GPL

Repository CRAN

Date/Publication 2021-03-01 07:10:07 UTC

NeedsCompilation no

R topics documented:

`ci.balance` *Experimental Function to Correct Confidence Intervals At or Near Boundaries of the Parameter Space by 'Sliding' the Interval on the Quantile Scale.*

Description

Experimental function to correct confidence intervals at or near boundaries of the parameter space by 'sliding' the interval on the quantile scale.

Usage

```
ci.balance(x, est, confidence=0.95, alpha=1-confidence, minval, maxval,
           na.rm=TRUE)
```

Arguments

`x`	Bootstrap parameter estimates.
`est`	Observed value of the parameter.
`confidence`	Confidence level for the interval. Defaults to 0.95.
`alpha`	Type I error rate (size) for the interval. Defaults to 1-`confidence`.
`minval`	A numeric value specifying the lower bound of the parameter space. Leave unspecified (the default) if there is no lower bound.
`maxval`	A numeric value specifying the upper bound of the parameter space. Leave unspecified (the default) if there is no upper bound.
`na.rm`	logical. Should missing values be removed?

Details

EXPERIMENTAL FUNCTION:

This function attempts to compute a proper conf*100% confidence interval for parameters at or near the boundary of the parameter space using bootstrapped parameter estimates by 'sliding' the confidence interval on the quantile scale.

This is accomplished by attempting to place a conf *100% interval symmetrically *on the quantile scale* about the observed value. If a symmetric interval would exceed the observed data at the upper (lower) end, a one-sided interval is computed with the upper (lower) boundary fixed at the the upper (lower) boundary of the parameter space.

Value

A list containing:

`ci` A 2-element vector containing the lower and upper confidence limits. The names of the elements of the vector give the actual quantile values used for the interval or one of the character strings "Upper Boundary" or "Lower Boundary".

`overflow.upper, overflow.lower`
The number of elements beyond those observed that would be needed to compute a symmetric (on the quantile scale) confidence interval.

`n.above, n.below`
The number of bootstrap values which are above (below) the observed value.

`lower.n, upper.n`
The index of the value used for the endpoint of the confidence interval or the character string "Upper Boundary" ("Lower Boundary").

Author(s)

Gregory R. Warnes <greg@warnes.net >

See Also

boot, bootstrap, Used by diseq.ci.

Examples

```
# These are nonsensical examples which simply exercise the
# computation. See the code to diseq.ci for a real example.
#
# FIXME: Add real example using boot or bootstrap.

set.seed(7981357)
x <- abs(rnorm(100,1))
ci.balance(x,1, minval=0)
ci.balance(x,1)

x <- rnorm(100,1)
x <- ifelse(x>1, 1, x)
ci.balance(x,1, maxval=1)
ci.balance(x,1)
```

Depreciated *Depreciated functions*

Description

These functions are depreciated.

Usage

```
power.casectrl(...)
```

Arguments

...	All arguments are ignored

Details

The power.casectl function contained serious errors. For some time, replacements were provided by the BioConductor GeneticsDesign package.

In specific, the power.casectl function used an expected contingency table to create the test statistic that was erroneously based on the underlying null, rather than on the marginal totals of the observed table. In addition, the modeling of dominant and recessive modes of inheritance had assumed a "perfect" genotype with no disease, whereas in reality a dominant or recessive mode of inheritance simply means that two of the genotypes will have an identical odds ratio compared to the 3rd genotype (the other homozygote).

diseq *Estimate or Compute Confidence Interval for the Single-Marker Disequilibrium*

Description

Estimate or compute confidence interval for single-marker disequilibrium.

Usage

```
diseq(x, ...)
## S3 method for class 'diseq'
print(x, show=c("D","D'","r","R^2","table"), ...)
diseq.ci(x, R=1000, conf=0.95, correct=TRUE, na.rm=TRUE, ...)
```

Arguments

x	genotype or haplotype object.
show	a character value or vector indicating which disequilibrium measures should be displayed. The default is to show all of the available measures. show="table" will display a table of observed, expected, and observed-expected frequencies.
conf	Confidence level to use when computing the confidence level for D-hat. Defaults to 0.95, should be in (0,1).
R	Number of bootstrap iterations to use when computing the confidence interval. Defaults to 1000.
correct	See details.
na.rm	logical. Should missing values be removed?
...	optional parameters passed to boot.ci (diseq.ci) or ignored.

Details

For a single-gene marker, diseq computes the Hardy-Weinberg (dis)equilibrium statistic D, D', r (the correlation coefficient), and r^2 for each pair of allele values, as well as an overall summary value for each measure across all alleles. print.diseq displays the contents of a diseq object. diseq.ci computes a bootstrap confidence interval for this estimate.

For consistency, I have applied the standard definitions for D, D', and r from the Linkage Disequilibrium case, replacing all marker probabilities with the appropriate allele probabilities.

Thus, for each allele pair,

- D is defined as the half of the raw difference in frequency between the observed number of heterozygotes and the expected number:

$$D = \frac{1}{2}(p_{ij} + p_{ji}) - p_i p_j$$

- D' rescales D to span the range [-1,1]

$$D' = \frac{D}{D_{max}}$$

  where, if D > 0:

$$D_{max} = \min p_i p_j, p_j p_i = p_i p_j$$

  or if D < 0:

$$D_{max} = \min p_i(1 - p_j), p_j(1 - p_i)$$

- r is the correlation coefficient between two alleles,
  and can be computed by

$$r = \frac{-D}{\sqrt{(p_i * (1 - p_i)p(j)(1 - p_j))}}$$

where

- - p_i defined as the observed probability of allele 'i',
- -p_j defined as the observed probability of allele 'j', and
- -p_{ij} defined as the observed probability of the allele pair 'ij'.

When there are more than two alleles, the summary values for these statistics are obtained by computing a weighted average of the absolute value of each allele pair, where the weight is determined by the expected frequency. For example:

$$D_{overall} = \sum_{i \neq j} |D_{ij}| * p_{ij}$$

Bootstrapping is used to generate confidence interval in order to avoid reliance on parametric assumptions, which will not hold for alleles with low frequencies (e.g. D' following a a Chi-square distribution).

See the function `HWE.test` for testing Hardy-Weinberg Equilibrium, $D = 0$.

Value

`diseq` returns an object of class `diseq` with components

- callfunction call used to create this object
- data2-way table of allele pair counts
- D.hatmatrix giving the observed count, expected count, observed - expected difference, and estimate of disequilibrium for each pair of alleles as well as an overall disequilibrium value.
- TODOmore slots to be documented

`diseq.ci` returns an object of class `boot.ci`

Author(s)

Gregory R. Warnes <greg@warnes.net >

See Also

`genotype`, `HWE.test`, `boot`, `boot.ci`

Examples

```
example.data   <- c("D/D","D/I","D/D","I/I","D/D",
                    "D/D","D/D","D/D","I/I","")
g1  <- genotype(example.data)
g1

diseq(g1)
diseq.ci(g1)
HWE.test(g1)  # does the same, plus tests D-hat=0

three.data   <- c(rep("A/A",8),
```

```
                  rep("C/A",20),
                  rep("C/T",20),
                  rep("C/C",10),
                  rep("T/T",3))

g3  <- genotype(three.data)
g3

diseq(g3)
diseq.ci(g3, ci.B=10000, ci.type="bca")

# only show observed vs expected table
print(diseq(g3),show='table')
```

`expectedGenotypes` *Construct expected genotypes/haplotypes according to known allele variants*

Description

`expectedGenotypes` constructs expected genotypes according to known allele variants, which can be quite tedious with large number of allele variants. It can handle different level of ploidy.

Usage

```
expectedGenotypes(x, alleles=allele.names(x), ploidy=2, sort=TRUE,
                  haplotype=FALSE)
expectedHaplotypes(x, alleles=allele.names(x), ploidy=2, sort=TRUE,
                   haplotype=TRUE)
```

Arguments

`x`	genotype or haplotype
`alleles`	character, vector of allele names
`ploidy`	numeric, number of chromosome sets i.e. 2 for human autosomal genes
`sort`	logical, sort genotypes according to order of alleles in `alleles` argument
`haplotype`	logical, construct haplotypes i.e. ordered genotype At least one of `x` or `alleles` must be given.

Details

`expectedHaplotypes()` just calls `expectedGenotypes()` with argument `haplotype=TRUE`.

Value

A character vector with genotype names as "alele1/alele2" for diploid example. Length of output is $(n * (n + 1))/2$ for genotype (unordered genotype) and $n * n$ for haplotype (ordered genotype) for n allele variants.

Author(s)

Gregor Gorjanc

See Also

allele.names, genotype

Examples

```
## On genotype
prp <- c("ARQ/ARQ", "ARQ/ARQ", "ARR/ARQ", "AHQ/ARQ", "ARQ/ARQ")
alleles <- c("ARR", "AHQ", "ARH", "ARQ", "VRR", "VRQ")
expectedGenotypes(as.genotype(prp))
expectedGenotypes(as.genotype(prp, alleles=alleles))
expectedGenotypes(as.genotype(prp, alleles=alleles, reorder="yes"))

## Only allele names
expectedGenotypes(alleles=alleles)
expectedGenotypes(alleles=alleles, ploidy=4)

## Haplotype
expectedHaplotypes(alleles=alleles)
expectedHaplotypes(alleles=alleles, ploidy=4)[1:20]
```

---

genotype *Genotype or Haplotype Objects.*

Description

genotype creates a genotype object.

haplotype creates a haplotype object.

is.genotype returns TRUE if x is of class genotype

is.haplotype returns TRUE if x is of class haplotype

as.genotype attempts to coerce its argument into an object of class genotype.

as.genotype.allele.count converts allele counts (0,1,2) into genotype pairs ("A/A", "A/B", "B/B").

as.haplotype attempts to coerce its argument into an object of class haplotype.

nallele returns the number of alleles in an object of class genotype.

Usage

```
genotype(a1, a2=NULL, alleles=NULL, sep="/", remove.spaces=TRUE,
         reorder = c("yes", "no", "default", "ascii", "freq"),
         allow.partial.missing=FALSE, locus=NULL,
         genotypeOrder=NULL)
```

```
  haplotype(a1, a2=NULL, alleles=NULL, sep="/", remove.spaces=TRUE,
            reorder="no", allow.partial.missing=FALSE, locus=NULL,
            genotypeOrder=NULL)

  is.genotype(x)

  is.haplotype(x)

  as.genotype(x, ...)

  ## S3 method for class 'allele.count'
as.genotype(x, alleles=c("A","B"), ... )

  as.haplotype(x, ...)

  ## S3 method for class 'genotype'
print(x, ...)

  nallele(x)
```

Arguments

x	either an object of class genotype or haplotype or an object to be converted to class genotype or haplotype.
a1,a2	vector(s) or matrix containing two alleles for each individual. See details, below.
alleles	names (and order if reorder="yes") of possible alleles.
sep	character separator or column number used to divide alleles when a1 is a vector of strings where each string holds both alleles. See below for details.
remove.spaces	logical indicating whether spaces and tabs will be removed from a1 and a2 before processing.
reorder	how should alleles within an individual be reordered. If reorder="no", use the order specified by the alleles parameter. If reorder="freq" or reorder="yes", sort alleles within each individual by observed frequency. If reorder="ascii", reorder alleles in ASCII order (alphabetical, with all upper case before lower case). The default value for genotype is "freq". The default value for haplotype is "no".
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.
locus	object of class locus, gene, or marker, holding information about the source of this genotype.
genotypeOrder	character, vector of genotype/haplotype names so that further functions can sort genotypes/haplotypes in wanted order
...	optional arguments

Details

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

The genotype class considers the stored alleles to be unordered, i.e., "C/T" is equivalent to "T/C". The haplotype class considers the order of the alleles to be significant so that "C/T" is distinct from "T/C".

When calling `genotype` or `haplotype`:

- If only `a1` is provided and is a character vector, it is assumed that each element encodes both alleles. In this case, if `sep` is a character string, `a1` 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.
- If `a1` is a matrix, it is assumed that column 1 contains allele 1 and column 2 contains allele 2.
- If `a1` and `a2` are both provided, each is assumed to contain one allele value so that the genotype for an individual is obtained by `paste(a1,a2,sep="/")`.

If `remove.spaces` is TRUE, (the default) any whitespace contained in `a1` and `a2` is removed when the genotypes are created. If whitespace is used as the separator, (eg "C C", "C T", ...), be sure to set remove.spaces to FALSE.

When the alleles are explicitly specified using the `alleles` argument, all potential alleles not present in the list will be converted to `NA`.

NOTE: `genotype` assumes that the order of the alleles is not important (E.G., "A/C" == "C/A"). Use class `haplotype` if order is significant.

If `genotypeOrder=NULL` (the default setting), then `expectedGenotypes` is used to get standard sorting order. Only unique values in `genotypeOrder` are used, which in turns means that the first occurrence prevails. When `genotypeOrder` is given some genotype names, but not all that appear in the data, the rest (those in the data and possible combinations based on allele variants) is automatically added at the end of `genotypeOrder`. This puts "missing" genotype names at the end of sort order. This feature is especially useful when there are a lot of allele variants and especially in haplotypes. See examples.

Value

The genotype class extends "factor" and haplotype extends genotype. Both classes have the following attributes:

`levels`	character vector of possible genotype/haplotype values stored coded by `paste( allele1, "/", allele2, sep="")`.
`allele.names`	character vector of possible alleles. For a SNP, these might be c("A","T"). For a variable length dinucleotyde repeat this might be c("136","138","140","148").
`allele.map`	matrix encoding how the factor levels correspond to alleles. See the source code to `allele.genotype()` for how to extract allele values using this matrix. Better yet, just use `allele.genotype()`.
`genotypeOrder`	character, genotype/haplotype names in defined order that can used for sorting in various functions. Note that this slot stores both ordered and unordered genotypes i.e. "A/B" and "B/A".

Author(s)

Gregory R. Warnes <greg@warnes.net> and Friedrich Leisch.

See Also

HWE.test, allele, homozygote, heterozygote, carrier, summary.genotype, allele.count, sort.genotype, genotypeOrder, locus, gene, marker, and %in% for default %in% method

Examples

```
# several examples of genotype data in different formats
example.data   <- c("D/D","D/I","D/D","I/I","D/D",
                    "D/D","D/D","D/D","I/I","")
g1  <- genotype(example.data)
g1

example.data2  <- c("C-C","C-T","C-C","T-T","C-C",
                    "C-C","C-C","C-C","T-T","")
g2  <- genotype(example.data2,sep="-")
g2


example.nosep  <- c("DD", "DI", "DD", "II", "DD",
                    "DD", "DD", "DD", "II", "")
g3  <- genotype(example.nosep,sep="")
g3

example.a1 <- c("D",  "D",  "D",  "I",  "D",  "D",  "D",  "D",  "I",  "")
example.a2 <- c("D",  "I",  "D",  "I",  "D",  "D",  "D",  "D",  "I",  "")
g4  <- genotype(example.a1,example.a2)
g4

example.mat <- cbind(a1=example.a1, a1=example.a2)
g5  <- genotype(example.mat)
g5

example.data5  <- c("D   /   D","D   /   I","D   /   D","I   /   I",
                    "D   /   D","D   /   D","D   /   D","D   /   D",
                    "I   /   I","")
g5  <- genotype(example.data5,rem=TRUE)
g5

# show how genotype and haplotype differ
data1 <- c("C/C", "C/T", "T/C")
data2 <- c("C/C", "T/C", "T/C")

test1  <- genotype( data1 )
test2  <- genotype( data2 )

test3  <-  haplotype( data1 )
test4  <-  haplotype( data2 )
```

```
test1==test2
test3==test4

test1=="C/T"
test1=="T/C"

test3=="C/T"
test3=="T/C"

## also
test1
test1
test3

test1
test1

test3
test3

## "Messy" example

m3  <-  c("D D/\t   D D","D\tD/   I",  "D D/   D D","I/   I",
          "D D/   D D","D D/   D D","D D/   D D","D D/   D D",
          "I/   I","/   ","/I")

genotype(m3)
summary(genotype(m3))

m4  <-  c("D D","D I","D D","I I",
          "D D","D D","D D","D D",
          "I I","   ","  I")

genotype(m4,sep=1)
genotype(m4,sep=" ",remove.spaces=FALSE)
summary(genotype(m4,sep=" ",remove.spaces=FALSE))

m5  <-  c("DD","DI","DD","II",
          "DD","DD","DD","DD",
          "II","   "," I")
genotype(m5,sep=1)
haplotype(m5,sep=1,remove.spaces=FALSE)

g5  <- genotype(m5,sep="")
h5  <- haplotype(m5,sep="")

heterozygote(g5)
homozygote(g5)
carrier(g5,"D")

g5[9:10]  <- haplotype(m4,sep=" ",remove=FALSE)[1:2]
g5
```

```
g5[9:10]
allele(g5[9:10],1)
allele(g5,1)[9:10]

# drop unused alleles
g5[9:10,drop=TRUE]
h5[9:10,drop=TRUE]

# Convert allele.counts into genotype

x <- c(0,1,2,1,1,2,NA,1,2,1,2,2,2)
g <- as.genotype.allele.count(x, alleles=c("C","T") )
g

# Use of genotypeOrder
example.data   <- c("D/D","D/I","I/D","I/I","D/D",
                    "D/D","D/I","I/D","I/I","")
summary(genotype(example.data))
genotypeOrder(genotype(example.data))

summary(genotype(example.data, genotypeOrder=c("D/D", "I/I", "D/I")))
summary(genotype(example.data, genotypeOrder=c(               "D/I")))
summary(haplotype(example.data, genotypeOrder=c(              "I/D", "D/I")))
example.data <- genotype(example.data)
genotypeOrder(example.data) <- c("D/D", "I/I", "D/I")
genotypeOrder(example.data)
```

`gregorius` *Probability of Observing All Alleles with a Given Frequency in a Sample of a Specified Size.*

Description

Probability of observing all alleles with a given frequency in a sample of a specified size.

Usage

```
gregorius(freq, N, missprob, tol = 1e-10, maxN = 10000, maxiter=100, showiter = FALSE)
```

Arguments

`freq`	(Minimum) Allele frequency (required)
`N`	Number of sampled genotypes
`missprob`	Desired maximum probability of failing to observe an allele.
`tol`	Omit computation for terms which contribute less than this value.
`maxN`	Largest value to consider when searching for N.
`maxiter`	Maximum number of iterations to use when searching for N.
`showiter`	Boolean flag indicating whether to show the iterations performed when searching for N.

Details

If freq and N are provided, but missprob is omitted, this function computes the probability of failing to observe all alleles with true underlying frequency freq when N diploid genotypes are sampled. This is accomplished using the sum provided in Corollary 2 of Gregorius (1980), omitting terms which contribute less than tol to the result.

When freq and missprob are provide, but N is omitted. A binary search on the range of [1,maxN] is performed to locate the smallest sample size, N, for which the probability of failing to observe all alleles with true underlying frequency freq is at most missprob. In this case, maxiter specifies the largest number of iterations to use in the binary search, and showiter controls whether the iterations of the search are displayed.

Value

A list containing the following values:

call	Function call used to generate this object.
method	One of the strings, "Compute missprob given N and freq", or "Determine minimal N given missprob and freq", indicating which type of computation was performed.
retval$freq	Specified allele frequency.
retval$N	Specified or computed sample size.
retval$missprob	Computed probability of failing to observe all of the alleles with frequency freq.

Note

This code produces sample sizes that are slightly larger than those given in table 1 of Gregorius (1980). This appears to be due to rounding of the computed missprobs by the authors of that paper.

Author(s)

Code submitted by David Duffy <davidD@qumr.edu.au>, substantially enhanced by Gregory R. Warnes <greg@warnes.net>.

References

Gregorius, H.R. 1980. The probability of losing an allele when diploid genotypes are sampled. Biometrics 36, 643-652.

Examples

```
# Compute the probability of missing an allele with frequency 0.15 when
# 20 genotypes are sampled:
gregorius(freq=0.15, N=20)

# Determine what sample size is required to observe all alleles with true
# frequency 0.15 with probability 0.95
gregorius(freq=0.15, missprob=1-0.95)
```

groupGenotype *Group genotype values*

Description

groupGenotype groups genotype or haplotype values according to given "grouping/mapping" information

Usage

```
groupGenotype(x, map, haplotype=FALSE, factor=TRUE, levels=NULL, verbose=FALSE)
```

Arguments

x	genotype or haplotype
map	list, mapping information, see details and examples
haplotype	logical, should values in a map be treated as haplotypes or genotypes, see details
factor	logical, should output be a factor or a character
levels	character, optional vector of level names if factor is produced (factor=TRUE); the default is to use the sort order of the group names in map
verbose	logical, print genotype names that match entries in the map - mainly used for debugging

Details

Examples show how map can be constructed. This are the main points to be aware of:

- names of list components are used as new group names
- list components hold genotype names per each group
- genotype names can be specified directly i.e. "A/B" or abbreviated such as "A/*" or even "*/*", where "*" matches any possible allele, but read also further on
- all genotype names that are not specified can be captured with ".else" (note the dot!)
- genotype names that were not specified (and ".else" was not used) are changed to NA

map is inspected before grouping of genotypes is being done. The following steps are done during inspection:

- ".else" must be at the end (if not, it is moved) to match everything that has not yet been defined
- any specifications like "A/*", "*/A", or "*/*" are extended to all possible genotypes based on alleles in argument alleles - in case of haplotype=FALSE, "A/*" and "*/A" match the same genotypes

- since use of "*" and ".else" can cause duplicates along the whole map, duplicates are removed sequentially (first occurrence is kept)

Using ".else" or "*/*" at the end of the map produces the same result, due to removing duplicates sequentially.

Value

A factor or character vector with genotypes grouped

Author(s)

Gregor Gorjanc

See Also

genotype, haplotype, factor, and levels

Examples

```
## --- Setup ---

x <- c("A/A", "A/B", "B/A", "A/C", "C/A", "A/D", "D/A",
       "B/B", "B/C", "C/B", "B/D", "D/B",
       "C/C", "C/D", "D/C",
       "D/D")
g <- genotype(x, reorder="yes")
## "A/A" "A/B" "A/B" "A/C" "A/C" "A/D" "A/D" "B/B" "B/C" "B/C" "B/D" "B/D"
## "C/C" "C/D" "C/D" "D/D"

h <- haplotype(x)
## "A/A" "A/B" "B/A" "A/C" "C/A" "A/D" "D/A" "B/B" "B/C" "C/B" "B/D" "D/B"
## "C/C" "C/D" "D/C" "D/D"

## --- Use of "A/A", "A/*" and ".else" ---

map <- list("homoG"=c("A/A", "B/B", "C/C", "D/D"),
            "heteroA*"=c("A/B", "A/C", "A/D"),
            "heteroB*"=c("B/*"),
            "heteroRest"=".else")

(tmpG <- groupGenotype(x=g, map=map, factor=FALSE))
(tmpH <- groupGenotype(x=h, map=map, factor=FALSE, haplotype=TRUE))

## Show difference between genotype and haplotype treatment
cbind(as.character(h), gen=tmpG, hap=tmpH, diff=!(tmpG == tmpH))
##              gen          hap          diff
##  [1,] "A/A" "homoG"      "homoG"      "FALSE"
##  [2,] "A/B" "heteroA*"   "heteroA*"   "FALSE"
##  [3,] "B/A" "heteroA*"   "heteroB*"   "TRUE"
##  [4,] "A/C" "heteroA*"   "heteroA*"   "FALSE"
```

```
##  [5,] "C/A" "heteroA*"   "heteroRest" "TRUE"
##  [6,] "A/D" "heteroA*"   "heteroA*"   "FALSE"
##  [7,] "D/A" "heteroA*"   "heteroRest" "TRUE"
##  [8,] "B/B" "homoG"      "homoG"      "FALSE"
##  [9,] "B/C" "heteroB*"   "heteroB*"   "FALSE"
## [10,] "C/B" "heteroB*"   "heteroRest" "TRUE"
## [11,] "B/D" "heteroB*"   "heteroB*"   "FALSE"
## [12,] "D/B" "heteroB*"   "heteroRest" "TRUE"
## [13,] "C/C" "homoG"      "homoG"      "FALSE"
## [14,] "C/D" "heteroRest" "heteroRest" "FALSE"
## [15,] "D/C" "heteroRest" "heteroRest" "FALSE"
## [16,] "D/D" "homoG"      "homoG"      "FALSE"

map <- list("withA"="A/*", "rest"=".else")
groupGenotype(x=g, map=map, factor=FALSE)
##  [1] "withA" "withA" "withA" "withA" "withA" "withA" "withA" "rest"  "rest"
## [10] "rest"  "rest"  "rest"  "rest"  "rest"  "rest"  "rest"

groupGenotype(x=h, map=map, factor=FALSE, haplotype=TRUE)
##  [1] "withA" "withA" "rest"  "withA" "rest"  "withA" "rest"  "rest"  "rest"
## [10] "rest"  "rest"  "rest"  "rest"  "rest"  "rest"  "rest"

## --- Use of "*/*" ---

map <- list("withA"="A/*", withB="*/*")
groupGenotype(x=g, map=map, factor=FALSE)
##  [1] "withA" "withA" "withA" "withA" "withA" "withA" "withA" "withB" "withB"
## [10] "withB" "withB" "withB" "withB" "withB" "withB" "withB"

## --- Missing genotype specifications produces NA's ---

map <- list("withA"="A/*", withB="B/*")
groupGenotype(x=g, map=map, factor=FALSE)
##  [1] "withA" "withA" "withA" "withA" "withA" "withA" "withA" "withB" "withB"
## [10] "withB" "withB" "withB" NA      NA      NA      NA

groupGenotype(x=h, map=map, factor=FALSE, haplotype=TRUE)
##  [1] "withA" "withA" "withB" "withA" NA      "withA" NA      "withB" "withB"
## [10] NA      "withB" NA      NA      NA      NA      NA
```

---

homozygote *Extract Features of Genotype objects*

Description

homozygote creates an vector of logicals that are true when the alleles of the corresponding observation are the identical.

heterozygote creates an vector of logicals that are true when the alleles of the corresponding observation differ.

carrier create a logical vector or matrix of logicals indicating whether the specified alleles are present.

allele.count returns the number of copies of the specified alleles carried by each observation.

allele extract the specified allele(s) as a character vector or a 2 column matrix.

allele.names extract the set of allele names.

Usage

```
homozygote(x,  allele.name, ...)
heterozygote(x, allele.name, ...)
carrier(x, allele.name, ...)
## S3 method for class 'genotype'
carrier(x, allele.name=allele.names(x),
        any=!missing(allele.name), na.rm=FALSE, ...)
allele.count(x, allele.name=allele.names(x),any=!missing(allele.name),
             na.rm=FALSE)
allele(x, which=c(1,2) )
allele.names(x)
```

Arguments

x	genotype object
...	optional parameters (ignored)
allele.name	character value or vector of allele names
any	logical value. When TRUE, a single count or indicator is returned by combining the results for all of the elements of allele. If FALSE separate counts or indicators should be returned for each element of allele. Defaults to FALSE if allele is missing. Otherwise defaults to TRUE.
na.rm	logical value indicating whether to remove missing values. When true, any NA values will be replaced by 0 or FALSE as appropriate. Defaults to FALSE.
which	selects which allele to return. For first allele use 1. For second allele use 2. For both (the default) use c(1,2).

Details

When the allele.name argument is given, heterozygote and homozygote return TRUE if *exactly* one or both alleles, respectively, match the specified allele.name.

Value

homozygote and heterozygote return a vector of logicals.

carrier returns a logical vector if only one allele is specified, or if any is TRUE. Otherwise, it returns matrix of logicals with one row for each element of allele.

allele.count returns a vector of counts if only one allele is specified, or if any is TRUE. Otherwise, it returns matrix of counts with one row for each element of allele.

`allele` returns a character vector when one allele is specified. When 2 alleles are specified, it returns a 2 column character matrix.

`allele.names` returns a character vector containing the set of allele names.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

genotype, HWE.test,

summary.genotype,

locus gene marker

Examples

```
example.data   <- c("D/D","D/I","D/D","I/I","D/D","D/D","D/D","D/D","I/I","")
g1  <- genotype(example.data)
g1

heterozygote(g1)
homozygote(g1)

carrier(g1,"D")
carrier(g1,"D",na.rm=TRUE)

# get count of one allele
allele.count(g1,"D")

# get count of each allele
allele.count(g1)  # equivalent to
allele.count(g1, c("D","I"), any=FALSE)

# get combined count for both alleles
allele.count(g1,c("I","D"))

# get second allele
allele(g1,2)

# get both alleles
allele(g1)
```

HWE.chisq *Perform Chi-Square Test for Hardy-Weinberg Equilibrium*

Description

Test the null hypothesis that Hardy-Weinberg equilibrium holds using the Chi-Square method.

Usage

```
HWE.chisq(x, ...)
## S3 method for class 'genotype'
HWE.chisq(x, simulate.p.value=TRUE, B=10000, ...)
```

Arguments

x	genotype or haplotype object.
simulate.p.value	a logical value indicating whether the p-value should be computed using simulation instead of using the χ^2 approximation. Defaults to TRUE.
B	Number of simulation iterations to use when simulate.p.value=TRUE. Defaults to 10000.
...	optional parameters passed to chisq.test

Details

This function generates a 2-way table of allele counts, then calls chisq.test to compute a p-value for Hardy-Weinberg Equilibrium. By default, it uses an unadjusted Chi-Square test statistic and computes the p-value using a simulation/permutation method. When simulate.p.value=FALSE, it computes the test statistic using the Yates continuity correction and tests it against the asymptotic Chi-Square distribution with the approproate degrees of freedom.

Note: The Yates continuty correction is applied *only* when simulate.p.value=FALSE, so that the reported test statistics when simulate.p.value=FALSE and simulate.p.value=TRUE will differ.

Value

An object of class htest.

See Also

HWE.exact, HWE.test, diseq, diseq.ci, allele, chisq.test, boot, boot.ci

Examples

```
example.data   <- c("D/D","D/I","D/D","I/I","D/D",
                    "D/D","D/D","D/D","I/I","")
g1  <- genotype(example.data)
g1

HWE.chisq(g1)
# compare with
HWE.exact(g1)
# and
HWE.test(g1)

three.data   <- c(rep("A/A",8),
                  rep("C/A",20),
                  rep("C/T",20),
                  rep("C/C",10),
                  rep("T/T",3))

g3  <- genotype(three.data)
g3

HWE.chisq(g3, B=10000)
```

`HWE.exact` *Exact Test of Hardy-Weinberg Equilibrium for 2-Allele Markers*

Description

Exact test of Hardy-Weinberg Equilibrium for 2 Allele Markers.

Usage

```
HWE.exact(x)
```

Arguments

`x`	Genotype object

Value

Object of class 'htest'.

Note

This function only works for genotypes with exactly 2 alleles.

Author(s)

David Duffy <davidD@qimr.edu.au> with modifications by Gregory R. Warnes <greg@warnes.net>

References

Emigh TH. (1980) "Comparison of tests for Hardy-Weinberg Equilibrium", Biometrics, 36, 627-642.

See Also

HWE.chisq, HWE.test, diseq, diseq.ci

Examples

```
example.data   <- c("D/D","D/I","D/D","I/I","D/D",
                    "D/D","D/D","D/D","I/I","")
g1  <- genotype(example.data)
g1

HWE.exact(g1)
# compare with
HWE.chisq(g1)



g2 <- genotype(sample( c("A","C"), 100, p=c(100,10), rep=TRUE),
               sample( c("A","C"), 100, p=c(100,10), rep=TRUE) )
HWE.exact(g2)
```

HWE.test *Estimate Disequilibrium and Test for Hardy-Weinberg Equilibrium*

Description

Estimate disequilibrium parameter and test the null hypothesis that Hardy-Weinberg equilibrium holds.

Usage

```
HWE.test(x, ...)
## S3 method for class 'genotype'
HWE.test(x, exact = nallele(x)==2, simulate.p.value=!exact,
         B=10000, conf=0.95, ci.B=1000, ... )
## S3 method for class 'data.frame'
HWE.test(x, ..., do.Allele.Freq=TRUE, do.HWE.test=TRUE)
## S3 method for class 'HWE.test'
print(x, show=c("D","D'","r","table"), ...)
```

Arguments

x	genotype or haplotype object.
exact	a logical value indicated whether the p-value should be computed using the exact method, which is only available for 2 allele genotypes.
simulate.p.value	a logical value indicating whether the p-value should be computed using simulation instead of using the χ^2 approximation. Defaults to TRUE.
B	Number of simulation iterations to use when simulate.p.value=TRUE. Defaults to 10000.
conf	Confidence level to use when computing the confidence level for D-hat. Defaults to 0.95, should be in (0,1).
ci.B	Number of bootstrap iterations to use when computing the confidence interval. Defaults to 1000.
show	a character vector containing the names of HWE test statistics to display from the set of "D", "D'", "r", and "table".
...	optional parameters passed to HWE.test (data.frame method) or chisq.test (base method).
do.Allele.Freq	logicial indication whether to summarize allele frequencies.
do.HWE.test	logicial indication whether to perform HWE tests

Details

HWE.test calls diseq to computes the Hardy-Weinberg (dis)equilibrium statistics D, D', and r (correlation coefficient). Next it calls diseq.ci to compute a bootstrap confidence interval for these estimates. Finally, it calls chisq.test to compute a p-value for Hardy-Weinberg Equilibrium using a simulation/permutation method.

Using bootstrapping for the confidence interval and simulation for the p-value avoids reliance on the assumptions the underlying Chi-square approximation. This is particularly important when some allele pairs have small counts.

For details on the definition of D, D', and r, see the help page for diseq.

Value

An object of class HWE.test with components

diseq	A diseq object providing details on the disequilibrium estimates.
ci	A diseq.ci object providing details on the bootstrap confidence intervals for the disequilibrium estimates.
test	A htest object providing details on the permutation based Chi-square test.
call	function call used to creat this object.
conf, B, ci.B, simulate.p.value	values used for these arguments.

Author(s)

Gregory R. Warnes <greg@warnes.net >

See Also

genotype, diseq, diseq.ci, HWE.chisq, HWE.exact, chisq.test

Examples

```
## Marker with two alleles:
example.data   <- c("D/D","D/I","D/D","I/I","D/D",
                    "D/D","D/D","D/D","I/I","")
g1  <- genotype(example.data)
g1

HWE.test(g1)

## Compare with individual calculations:
diseq(g1)
diseq.ci(g1)
HWE.chisq(g1)
HWE.exact(g1)


## Marker with three alleles: A, C, and T
three.data   <- c(rep("A/A",16),
                  rep("C/A",40),
                  rep("C/T",40),
                  rep("C/C",20),
                  rep("T/T",6))

g3  <- genotype(three.data)
g3

HWE.test(g3, ci.B=10000)
```

LD *Pairwise linkage disequilibrium between genetic markers.*

Description

Compute pairwise linkage disequilibrium between genetic markers

Usage

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

```
## S3 method for class 'data.frame'
LD(g1,...)
```

Arguments

g1	genotype object or dataframe containing genotype objects
g2	genotype object (ignored if g1 is a dataframe)
...	optional arguments (ignored)

Details

Linkage disequilibrium (LD) is the non-random association of marker alleles and can arise from marker proximity or from selection bias.

`LD.genotype` estimates the extent of LD for a single pair of genotypes. `LD.data.frame` computes LD for all pairs of genotypes contained in a data frame. Before starting, `LD.data.frame` checks the class and number of alleles of each variable in the dataframe. If the data frame contains non-genotype objects or genotypes with more or less than 2 alleles, these will be omitted from the computation and a warning will be generated.

Three estimators of LD are computed:

- D raw difference in frequency between the observed number of AB pairs and the expected number:

$$D = p_{AB} - p_A p_B$$

- D' scaled D spanning the range [-1,1]

$$D' = \frac{D}{D_{max}}$$

where, if D > 0:

$$D_{max} = \min(p_A p_b, p_a p_B)$$

or if D < 0:

$$D_{max} = \max -p_A p_B, -p_a p_b$$

- r correlation coefficient between the markers

$$r = \frac{-D}{\sqrt{(p_A * p_a * p_B * p_b)}}$$

where

- - p_A is defined as the observed probability of allele 'A' for marker 1,
- - $p_a = 1 - p_A$ is defined as the observed probability of allele 'a' for marker 1,
- -p_B is defined as the observed probability of allele 'B' for marker 2, and
- -$p_b = 1 - p_B$ is defined as the observed probability of allele 'b' for marker 2, and
- -p_{AB} is defined as the probability of the marker allele pair 'AB'.

For genotype data, AB/ab cannot be distinguished from aB/Ab. Consequently, we estimate p_{AB} using maximum likelihood and use this value in the computations.

Value

LD.genotype returns a 5 element list:

call	the matched call
D	Linkage disequilibrium estimate
Dprime	Scaled linkage disequilibrium estimate
corr	Correlation coefficient
nobs	Number of observations
chisq	Chi-square statistic for linkage equilibrium (i.e., D=D'=corr=0)
p.value	Chi-square p-value for marker independence

LD.data.frame returns a list with the same elements, but each element is a matrix where the upper off-diagonal elements contain the estimate for the corresponding pair of markers. The other matrix elements are NA.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

genotype, HWE.test

Examples

```
g1 <- genotype( c('T/A',    NA, 'T/T',    NA, 'T/A',    NA, 'T/T', 'T/A',
                  'T/T', 'T/T', 'T/A', 'A/A', 'T/T', 'T/A', 'T/A', 'T/T',
                     NA, 'T/A', 'T/A',   NA) )

g2 <- genotype( c('C/A', 'C/A', 'C/C', 'C/A', 'C/C', 'C/A', 'C/A', 'C/A',
                  'C/A', 'C/C', 'C/A', 'A/A', 'C/A', 'A/A', 'C/A', 'C/C',
                  'C/A', 'C/A', 'C/A', 'A/A') )


g3 <- genotype( c('T/A', 'T/A', 'T/T', 'T/A', 'T/T', 'T/A', 'T/A', 'T/A',
                  'T/A', 'T/T', 'T/A', 'T/T', 'T/A', 'T/A', 'T/A', 'T/T',
                  'T/A', 'T/A', 'T/A', 'T/T') )

# Compute LD on a single pair

LD(g1,g2)

# Compute LD table for all 3 genotypes

data <- makeGenotypes(data.frame(g1,g2,g3))
LD(data)
```

locus *Create and Manipulate Locus, Gene, and Marker Objects*

Description

locus, gene, and marker create objects to store information, respectively, about genetic loci, genes, and markers.

is.locus, is.gene, and ismarker test whether an object is a member of the respective class.

as.character.locus, as.character.gene, as.character.marker return a character string containing a compact encoding the object.

getlocus, getgene, getmarker extract locus data (if present) from another object.

locus<-, marker<-, and gene<- adds locus data to an object.

Usage

```
  locus(name, chromosome, arm=c("p", "q", "long", "short", NA),
        index.start, index.end=NULL)

  gene(name, chromosome, arm=c("p", "q", "long", "short"),
       index.start, index.end=NULL)

  marker(name, type, locus.name, bp.start, bp.end = NULL,
         relative.to = NULL, ...)

  is.locus(x)

  is.gene(x)

  is.marker(x)

  ## S3 method for class 'locus'
as.character(x, ...)

  ## S3 method for class 'gene'
as.character(x, ...)

  ## S3 method for class 'marker'
as.character(x, ...)

  getlocus(x, ...)

  locus(x) <- value

  marker(x) <- value
```

```
gene(x) <- value
```

Arguments

name	character string giving locus, gene, or marker name
chromosome	integer specifying chromosome number (1:23 for humans).
arm	character indicating long or short arm of the chromosome. Long is be specified by "long" or "p". Short is specified by "short" or "q".
index.start	integer specifying location of start of locus or gene on the chromosome.
index.end	optional integer specifying location of end of locus or gene on the chromosome.
type	character string indicating marker type, e.g. "SNP"
locus.name	either a character string giving the name of the locus or gene (other details may be specified using ...) or a locus or gene object.
bp.start	start location of marker, in base pairs
bp.end	end location of marker, in base pairs (optional)
relative.to	location (optional) from which bp.start and bp.end are calculated.
...	parameters for locus used to fill in additional details on the locus or gene within which the marker is located.
x	an object of class locus, gene, or marker, or (for getlocus, locus<-, marker<-, and gene<-) an object that may contain a locus attribute or field, notably a genotype object.
value	locus, marker, or gene object

Value

Object of class locus and geneare lists with the elements:

name	character string giving locus, gene, or marker name
chromosome	integer specifying chromosome number (1:23 for humans).
arm	character indicating long or short arm of the chromosome. Long is be specified by "long" or "p". Short is specified by "short" or "q".
index.start	integer specifying location of start of locus or gene on the chromosome.
index.end	optional integer specifying location of end of locus or gene on the chromosome.

Objects of class marker add the additional fields:

marker.name	character string giving the name of the marker
bp.start	start location of marker, in base pairs
bp.end	end location of marker, in base pairs (optional)
relative.to	location (optional) from which bp.start and bp.end are calculated.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

genotype,

Examples

```
ar2  <- gene("AR2",chromosome=7,arm="q",index.start=35)
ar2

par  <- locus(name="AR2 Psedogene",
              chromosome=1,
              arm="q",
              index.start=32,
              index.end=42)
par

c109t  <- marker(name="C-109T",
                 type="SNP",
                 locus.name="AR2",
                 chromosome=7,
                 arm="q",
                 index.start=35,
                 bp.start=-109,
                 relative.to="start of coding region")
c109t

c109t  <- marker(name="C-109T",
                 type="SNP",
                 locus=ar2,
                 bp.start=-109,
                 relative.to="start of coding region")
c109t




example.data   <- c("D/D","D/I","D/D","I/I","D/D",
                    "D/D","D/D","D/D","I/I","")
g1  <- genotype(example.data, locus=ar2)
g1

getlocus(g1)

summary(g1)
HWE.test(g1)

g2  <- genotype(example.data, locus=c109t)
summary(g2)

getlocus(g2)

heterozygote(g2)
homozygote(g1)
```

```
allele(g1,1)

carrier(g1,"I")

heterozygote(g2)
```

`makeGenotypes` *Convert columns in a dataframe to genotypes or haplotypes*

Description

Convert columns in a dataframe to genotypes or haplotypes.

Usage

```
makeGenotypes(data, convert, sep = "/", tol = 0.5, ..., method=as.genotype)
makeHaplotypes(data, convert, sep = "/", tol = 0.9, ...)
```

Arguments

`data`	Dataframe containing columns to be converted
`convert`	Vector or list of pairs specifying which columns contain genotype/haplotype data. See below for details.
`sep`	Genotype separator
`tol`	See below.
`...`	Optional arguments to as.genotype function
`method`	Function used to perform the conversion.

Details

The functions makeGenotypes and makeHaplotypes allow the conversion of all of the genetic variables in a dataset to genotypes or haplotypes in a single step.

The parameter `convert` may be missing, a vector of column names, indexes or true/false indictators, or a list of column name or index pairs.

When the argument `convert` is not provided, the function will look for columns where at least `tol`*100% of the records contain the separator character `sep` ('/' by default). These columns will then be assumed to contain both of the genotype/haplotype alleles and will be converted in-place to genotype variables.

When the argument `convert` is a vector of column names, indexes or true/false indictators, the corresponding columns will be assumed to contain both of the genotype/haplotype alleles and will be converted in-place to genotype variables.

When the argument `convert` is a list containing column name or index pairs, the two elements of each pair will be assumed to contain the individual alleles of a genotype/haplotype. The first

column specified in each pair will be replaced with the new genotype/haplotype variable named name1 + sep + name2. The second column will be removed.

Note that the method argument may be used to supply a non-standard conversion function, such as as.genotype.allele.count, which converts from [0,1,2] to ['A/A','A/B','A/C'] (or the specified allele names). See the example below.

Value

Dataframe containing converted genotype/haplotype variables. All other variables will be unchanged.

Author(s)

Gregory R. Warnes <greg@warnes.net >

See Also

genotype

Examples

```
## Not run:
# common case
data <- read.csv(file="genotype_data.csv")
data <- makeGenotypes(data)

## End(Not run)

# Create a test data set where there are several genotypes in columns
# of the form "A/T".
test1 <- data.frame(Tmt=sample(c("Control","Trt1","Trt2"),20, replace=TRUE),
                G1=sample(c("A/T","T/T","T/A",NA),20, replace=TRUE),
                N1=rnorm(20),
                I1=sample(1:100,20,replace=TRUE),
                G2=paste(sample(c("134","138","140","142","146"),20,
                                replace=TRUE),
                         sample(c("134","138","140","142","146"),20,
                                replace=TRUE),
                         sep=" / "),
                G3=sample(c("A /T","T /T","T /A"),20, replace=TRUE),
                comment=sample(c("Possible Bad Data/Lab Error",""),20,
                               rep=TRUE)
                )
test1

# now automatically convert genotype columns
geno1 <- makeGenotypes(test1)
geno1

# Create a test data set where there are several haplotypes with alleles
# in adjacent columns.
test2 <- data.frame(Tmt=sample(c("Control","Trt1","Trt2"),20, replace=TRUE),
```

```
                    G1.1=sample(c("A","T",NA),20, replace=TRUE),
                    G1.2=sample(c("A","T",NA),20, replace=TRUE),
                    N1=rnorm(20),
                    I1=sample(1:100,20,replace=TRUE),
                    G2.1=sample(c("134","138","140","142","146"),20,
                                replace=TRUE),
                    G2.2=sample(c("134","138","140","142","146"),20,
                                replace=TRUE),
                    G3.1=sample(c("A ","T ","T "),20, replace=TRUE),
                    G3.2=sample(c("A ","T ","T "),20, replace=TRUE),
                    comment=sample(c("Possible Bad Data/Lab Error",""),20,
                                   rep=TRUE)
                   )
test2

# specifly the locations of the columns to be paired for haplotypes
makeHaplotypes(test2, convert=list(c("G1.1","G1.2"),6:7,8:9))

# Create a test data set where the data is coded as numeric allele
# counts (0-2).
test3 <- data.frame(Tmt=sample(c("Control","Trt1","Trt2"),20, replace=TRUE),
                    G1=sample(c(0:2,NA),20, replace=TRUE),
                    N1=rnorm(20),
                    I1=sample(1:100,20,replace=TRUE),
                    G2=sample(0:2,20, replace=TRUE),
                    comment=sample(c("Possible Bad Data/Lab Error",""),20,
                                   rep=TRUE)
                   )
test3

# specifly the locations of the columns, and a non-standard conversion
makeGenotypes(test3, convert=c('G1','G2'), method=as.genotype.allele.count)
```

`order.genotype` *Order/sort genotype/haplotype object*

Description

Order/sort genotype or haplotype object according to order of allele names or genotypes

Usage

```
## S3 method for class 'genotype'
order(..., na.last=TRUE, decreasing=FALSE,
  alleleOrder=allele.names(x), genotypeOrder=NULL)

## S3 method for class 'genotype'
```

```
sort(x, decreasing=FALSE, na.last=NA, ...,
  alleleOrder=allele.names(x), genotypeOrder=NULL)

genotypeOrder(x)
genotypeOrder(x) <- value
```

Arguments

...	genotype or haplotype in order method; not used for sort method
x	genotype or haplotype in sort method
na.last	as in default order or sort
decreasing	as in default order or sort
alleleOrder	character, vector of allele names in wanted order
genotypeOrder	character, vector of genotype/haplotype names in wanted order
value	the same as in argument order.genotype

Details

Argument genotypeOrder can be usefull, when you want that some genotypes appear "together", whereas they are not "together" by allele order.

Both methods (order and sort) work with genotype and haplotype classes.

If alleleOrder is given, genotypeOrder has no effect.

Genotypes/haplotypes, with missing alleles in alleleOrder are treated as NA and ordered according to order arguments related to NA values. In such cases a warning is issued ("Found data values not matching specified alleles. Converting to NA.") and can be safely ignored. Genotypes present in x, but not specified in genotypeOrder, are also treated as NA.

Value of genotypeOrder such as "B/A" matches also "A/B" in case of genotypes.

Only unique values in argument alleleOrder or genotypeOrder are used i.e. first occurrence prevails.

Value

The same as in order or sort

Author(s)

Gregor Gorjanc

See Also

genotype, allele.names, order, and sort

Examples

```
x <- c("C/C", "A/C", "A/A", NA, "C/B", "B/A", "B/B", "B/C", "A/C")
alleles <- c("A", "B", "C")

g <- genotype(x, alleles=alleles, reorder="yes")
## "C/C" "A/C" "A/A" NA    "B/C" "A/B" "B/B" "B/C" "A/C"

h <- haplotype(x, alleles=alleles)
## "C/C" "A/C" "A/A" NA    "C/B" "B/A" "B/B" "B/C" "A/C"

## --- Standard usage ---

sort(g)
## "A/A" "A/B" "A/C" "A/C" "B/B" "B/C" "B/C" "C/C" NA

sort(h)
## "A/A" "A/C" "A/C" "B/A" "B/B" "B/C" "C/B" "C/C" NA

## --- Reversed order of alleles ---

sort(g, alleleOrder=c("B", "C", "A"))
## "B/B" "B/C" "B/C" "A/B" "C/C" "A/C" "A/C" "A/A" NA
## note that A/B comes after B/C since it is treated as B/A;
## order of alleles (not in alleleOrder!) does not matter for a genotype

sort(h, alleleOrder=c("B", "C", "A"))
## "B/B" "B/C" "B/A" "C/B" "C/C" "A/C" "A/C" "A/A" NA

## --- Missing allele(s) in alleleOrder ---

sort(g, alleleOrder=c("B", "C"))
## "B/B" "B/C" "B/C" "C/C" "A/C" "A/A" NA    "A/B" "A/C"

sort(g, alleleOrder=c("B"))
## "B/B" "C/C" "A/C" "A/A" NA    "B/C" "A/B" "B/C" "A/C"
## genotypes with missing allele are treated as NA

sort(h, alleleOrder=c("B", "C"))
## "B/B" "B/C" "C/B" "C/C" "A/C" "A/A" NA    "B/A" "A/C"

sort(h, alleleOrder=c("B"))
## "B/B" "C/C" "A/C" "A/A" NA    "C/B" "B/A" "B/C" "A/C"

## --- Use of genotypeOrder ---

sort(g, genotypeOrder=c("A/A", "C/C", "B/B", "A/B", "A/C", "B/C"))
## "A/A" "C/C" "B/B" "A/B" "A/C" "A/C" "B/C" "B/C" NA

sort(h, genotypeOrder=c("A/A", "C/C", "B/B",
                        "A/C", "C/B", "B/A", "B/C"))
## "A/A" "C/C" "B/B" "A/C" "A/C" "C/B" "B/A" "B/C" NA
```

```
## --- Missing genotype(s) in genotypeOrder ---

sort(g, genotypeOrder=c(       "C/C",        "A/B", "A/C", "B/C"))
## "C/C" "A/B" "A/C" "A/C" "B/C" "B/C" "A/A" NA    "B/B"

sort(h, genotypeOrder=c(       "C/C",        "A/B", "A/C", "B/C"))
## "C/C" "A/C" "A/C" "B/C" "A/A" NA    "C/B" "B/A" "B/B"
```

`plot.genotype` *Plot genotype object*

Description

`plot.genotype` can plot genotype or allele frequency of a genotype object.

Usage

```
## S3 method for class 'genotype'
plot(x, type=c("genotype", "allele"),
  what=c("percentage", "number"), ...)
```

Arguments

`x`	genotype object, as genotype.
`type`	plot "genotype" or "allele" frequency, as character.
`what`	show "percentage" or "number", as character
`...`	Optional arguments for `barplot`.

Value

The same as in `barplot`.

Author(s)

Gregor Gorjanc

See Also

`genotype`, `barplot`

Examples

```
set <- c("A/A", "A/B", "A/B", "B/B", "B/B", "B/B",
         "B/B", "B/C", "C/C", "C/C")
set <- genotype(set, alleles=c("A", "B", "C"), reorder="yes")
plot(set)
plot(set, type="allele", what="number")
```

print.LD *Textual and graphical display of linkage disequilibrium (LD) objects*

Description

Textual and graphical display of linkage disequilibrium (LD) objects

Usage

```
## S3 method for class 'LD'
print(x, digits = getOption("digits"), ...)
## S3 method for class 'LD.data.frame'
print(x, ...)

## S3 method for class 'data.frame'
summary.LD(object, digits = getOption("digits"),
                     which = c("D", "D'", "r", "X^2", "P-value", "n", " "),
                     rowsep, show.all = FALSE, ...)
## S3 method for class 'summary.LD.data.frame'
print(x, digits = getOption("digits"), ...)

## S3 method for class 'LD.data.frame'
plot(x,digits=3, colorcut=c(0,0.01, 0.025, 0.5, 0.1, 1),
                   colors=heat.colors(length(colorcut)), textcol="black",
                   marker, which="D'", distance,  ...)


LDtable(x, colorcut=c(0,0.01, 0.025, 0.5, 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, ...)

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

Arguments

x,object	LD or LD.data.frame object
digits	Number of significant digits to display
which	Name(s) of LD information items to be displayed
rowsep	Separator between rows of data, use NULL for no separator.
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)

Gregory R. Warnes <greg@warnes.net>

See Also

LD, genotype, HWE.test

Examples

```
g1 <- genotype( c('T/A',    NA, 'T/T',    NA, 'T/A',    NA, 'T/T', 'T/A',
                  'T/T', 'T/T', 'T/A', 'A/A', 'T/T', 'T/A', 'T/A', 'T/T',
                     NA, 'T/A', 'T/A',   NA) )

g2 <- genotype( c('C/A', 'C/A', 'C/C', 'C/A', 'C/C', 'C/A', 'C/A', 'C/A',
                  'C/A', 'C/C', 'C/A', 'A/A', 'C/A', 'A/A', 'C/A', 'C/C',
                  'C/A', 'C/A', 'C/A', 'A/A') )


g3 <- genotype( c('T/A', 'T/A', 'T/T', 'T/A', 'T/T', 'T/A', 'T/A', 'T/A',
                  'T/A', 'T/T', 'T/A', 'T/T', 'T/A', 'T/A', 'T/A', 'T/T',
                  'T/A', 'T/A', 'T/A', 'T/T') )
data <- makeGenotypes(data.frame(g1,g2,g3))

# Compute & display  LD for one marker pair
ld <- LD(g1,g2)
print(ld)

# Compute LD table for all 3 genotypes
ldt <- LD(data)

# display the results
print(ldt)                               # textual display
LDtable(ldt)                            # graphical color-coded table
```

```
LDplot(ldt, distance=c(124, 834, 927))  # LD plot vs distance

# more markers makes prettier plots!
data <- list()
nobs <- 1000
ngene <- 20
s <- seq(0,1,length=ngene)
a1 <- a2 <- matrix("", nrow=nobs, ncol=ngene)
for(i in 1:length(s) )
{

  rallele <- function(p) sample( c("A","T"), 1, p=c(p, 1-p))

  if(i==1)
    {
      a1[,i] <- sample( c("A","T"), 1000, p=c(0.5,0.5), replace=TRUE)
      a2[,i] <- sample( c("A","T"), 1000, p=c(0.5,0.5), replace=TRUE)
    }
  else
    {
      p1 <- pmax( pmin( 0.25 + s[i] * as.numeric(a1[,i-1]=="A"),1 ), 0 )
      p2 <- pmax( pmin( 0.25 + s[i] * as.numeric(a2[,i-1]=="A"),1 ), 0 )
      a1[,i] <- sapply(p1, rallele )
      a2[,i] <- sapply(p2, rallele )
    }

  data[[paste("G",i,sep="")]] <- genotype(a1[,i],a2[,i])
}
data <- data.frame(data)
data <- makeGenotypes(data)

ldt <- LD(data)
plot(ldt, digits=2, marker=19) # do LDtable & LDplot on in a single
                               # graphics window
```

`summary.genotype` *Allele and Genotype Frequency from a Genotype or Haplotype Object*

Description

`summary.genotype` creates an object containing allele and genotype frequency from a `genotype` or `haplotype` object. `print.summary.genotype` displays a `summary.genotype` object.

Usage

```
  ## S3 method for class 'genotype'
summary(object, ..., maxsum)
  ## S3 method for class 'summary.genotype'
print(x,...,round=2)
```

Arguments

object, x	an object of class genotype or haplotype (for summary.genotype) or an object of class summary.genotype (for print.summary.genotype)
...	optional parameters. Ignored by summary.genotype, passed to print.matrix by print.summary,genotype.
maxsum	specifying any value for the parameter maxsum will cause summary.genotype to fall back to summary.factor.
round	number of digits to use when displaying proportions.

Details

Specifying any value for the parameter maxsum will cause fallback to summary.factor. This is so that the function summary.dataframe will give reasonable output when it contains a genotype column. (Hopefully we can figure out something better to do in this case.)

Value

The returned value of summary.genotype is an object of class summary.genotype which is a list with the following components:

locus	locus information field (if present) from x
.	
allele.names	vector of allele names
allele.freq	A two column matrix with one row for each allele, plus one row for NA values (if present). The first column, Count, contains the frequency of the corresponding allele value. The second column, Proportion, contains the fraction of alleles with the corresponding allele value. Note each observation contains two alleles, thus the Count field sums to twice the number of observations.
genotype.freq	A two column matrix with one row for each genotype, plus one row for NA values (if present). The first column, Count, contains the frequency of the corresponding genotype. The second column, Proportion, contains the fraction of genotypes with the corresponding value.

print.summary.genotype silently returns the object x.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

genotype, HWE.test, allele, homozygote, heterozygote, carrier,

allele.count locus gene marker

Examples

```
example.data   <- c("D/D","D/I","D/D","I/I","D/D",
                    "D/D","D/D","D/D","I/I","")
g1  <- genotype(example.data)
g1

summary(g1)
```

`undocumented` *Undocumented functions*

Description

These functions are undocumented. Some are internal and not intended for direct use. Some are not yet ready for end users. Others simply haven't been documented yet.

Author(s)

Gregory R. Warnes

`write.pop.file` *Create genetics data files*

Description

`write.pop.file` creates a 'pop' data file, as used by the GenePop (https://genepop.curtin.edu.au/) and LinkDos (https://genepop.curtin.edu.au/linkC.html) software packages.

`write.pedigree.file` creates a 'pedigree' data file, as used by the QTDT software package (http://csg.sph.umich.edu//abecasis/QTDT/).

`write.marker.file` creates a 'marker' data file, as used by the QTDT software package (http://csg.sph.umich.edu//abecasis/QTDT/).

Usage

```
write.pop.file(data, file = "", digits = 2, description = "Data from R")
write.pedigree.file(data, family, pid, father, mother, sex,
                    file="pedigree.txt")
write.marker.file(data, location, file="marker.txt")
```

Arguments

data	Data frame containing genotype objects to be exported
file	Output filename
digits	Number of digits to use in numbering genotypes, either 2 or 3.
description	Description to use as the first line of the 'pop' file.
family, pid, father, mother	Vector of family, individual, father, and mother id's, respectively.
sex	Vector giving the sex of the individual (1=Make, 2=Female)
location	Location of the marker relative to the gene of interest, in base pairs.

Details

The format of 'Pop' files is documented at https://genepop.curtin.edu.au/help_input.html, the format of 'pedigree' files is documented at http://csg.sph.umich.edu/abecasis/GOLD/docs/pedigree.html and the format of 'marker' files is documented at http://csg.sph.umich.edu/abecasis/GOLD/docs/map.html.

Value

No return value.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

write.table

Examples

```
# TBA
```

Index