Package ‘rres’

October 14, 2022

Type Package

Title Realized Relatedness Estimation and Simulation

Version 1.1

Date 2018-03-27

Author Bowen Wang

Maintainer Bowen Wang <bowenwang7@gmail.com>

Description Functions for studying realized genetic relatedness between people. Users will be able to simulate inheritance patterns given pedigree structures, generate SNP marker data given inheritance patterns, and estimate realized relatedness between pairs of individuals using SNP marker data. See Wang (2017) <doi:10.1534/genetics.116.197004>. This work was supported by National Institutes of Health grants R37 GM-046255.

License GPL (>= 2)

Imports Rcpp (>= 0.12.16), kernlab

LinkingTo Rcpp

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-06-04 21:54:23 UTC

R topics documented:

check.pedinfo ................................................................. 2
fgl2ibd ................................................................. 2
fgl2relatedness .......................................................... 3
get.pedindex ............................................................... 4
grm.matrix ............................................................... 4
grm.pair ................................................................. 5
ibd.length ............................................................... 7
ibd.marker ............................................................... 8
ibd.proportion ........................................................... 9
check.pedinfo

Description

check.pedinfo checks that the pedigree information provided is consistent.

Usage

check.pedinfo(pedinfo)

Arguments

pedinfo dataframe.

Details

Member ID must be unique. Parents must precede offsprings. Sex information must match parental status, and are coded 1 and 2 for male and female respectively. An error message will be produced only if inconsistencies are found.

fgl2ibd

Description

fgl2ibd determines pairwise IBD state given the four founder genome labels of two individuals at a marker.

Usage

fgl2ibd(fgl1p, fgl1m, fgl2p, fgl2m)
**fgl2relatedness**

**Arguments**

- fgl1p, fgl1m, fgl2p, fgl2m
  - positive integer, represents founder genome label.

**Details**

IBD states take value from 1 to 15, which represent the indices of the underlying IBD states from 1111 to 1234 in lexicographical order. E.g., output 1 means IBD state 1111, output 2 means IBD state 1112 etc. Recoding in, e.g., Jacquard order, can be obtained using `recode.ibd`.

**Value**

A value between 1 and 15 representing index of IBD state in lexicographical order.

**Examples**

```
fgl2ibd(1, 1, 1, 1)
fgl2ibd(1, 2, 1, 2)
fgl2ibd(3, 4, 5, 6)
fgl2ibd(4, 5, 4, 4)
```

---

**fgl2relatedness**

*Score pairwise relatedness.*

**Description**

fgl2relatedness determines pairwise relatedness given the four founder genome labels of two individuals at a marker.

**Usage**

```
fgl2relatedness(fgl1p, fgl1m, fgl2p, fgl2m)
```

**Arguments**

- fgl1p, fgl1m, fgl2p, fgl2m
  - positive integer, represents founder genome label.

**Value**

A value in [0, 0.5, 1, 2] representing local relatedness coefficient.

**Examples**

```
fgl2relatedness(1, 1, 1, 1)
fgl2relatedness(1, 2, 1, 2)
fgl2relatedness(1, 2, 1, 3)
fgl2relatedness(3, 4, 5, 6)
fgl2relatedness(4, 5, 4, 4)
```
get.pedindex

Get pedigree index.

Description

get.pedindex returns indices of individuals in the pedigree.

Usage

get.pedindex(pedinfo, member.set)

Arguments

pedinfo dataframe.
member.set character vector.

Details

member.set contains member IDs of individuals of interest.

Value

An integer vector of indices for each individual of interest found in pedinfo.

Examples

# a simple pedigree with sibling marriage
pedigree = as.character(rep(1, 5))
member = as.character(c(11, 12, 21, 22, 31))
sex = as.numeric(c(1, 2, 1, 2, 1))
father = as.character(c(NA, NA, 11, 11, 21))
mother = as.character(c(NA, NA, 12, 12, 22))
pedinfo = data.frame(pedigree, member, sex, father, mother, stringsAsFactors = FALSE)
get.pedindex(pedinfo, c("22", "31"))

grm.matrix

GRM for multiple individuals

Description

grm.matrix computes relatedness estimates between every pairs of individuals.

Usage

grm.matrix(genotype, freq, method = "twostep", weights = NULL, init.est = NULL)
Arguments

- **genotype**: numeric matrix.
- **freq**: numeric vector, values between 0 and 1.
- **method**: string.
- **weights**: numeric vector, values between 0 and 1.
- **init.est**: numeric.

Details

genotype is the matrix of counts of reference alleles. Rows represents subjects and columns represents SNP markers. freq is the vector of reference allele frequencies.

The default method is "twostep", other options include "classic", "robust" and "general". When using the default "twostep" method, user can supply an initial estimate through init.est to bypass the first step. When "general" is selected, weights must also be specified. The difference between the two-step GRM, classic GRM and robust GRM is discussed in Wang et al. (2017).

References


See Also

- grm.pair.

grm.pair  
**GRM for a pair of individuals.**

Description

grm.pair computes relatedness estimates between two individuals.

Usage

```r
grm.pair(geno1, geno2, freq, method = "twostep", weights = NULL, init.est = NULL)
```

Arguments

- **geno1, geno2**: numeric vector.
- **freq**: numeric vector, values between 0 and 1.
- **method**: string.
- **weights**: numeric vector, values between 0 and 1.
- **init.est**: numeric.
Details

gen1 and gen2 are vectors of counts of reference alleles. freq is the vector of reference allele frequencies.

The default method is "twostep", other options include "classic", "robust" and "general". When using the default "twostep" method, user can supply an initial estimate through init.est to bypass the first step. When "general" is selected, weights must also be specified. The difference between the two-step GRM, classic GRM and robust GRM is discussed in Wang et al. (2017).

Value

An estimate of realized relatedness.

References


See Also

grm.matrix

Examples

# simulate genotypes for a full sib pair
pedigree = as.character(rep(1, 4))
member = as.character(c(11, 12, 21, 22))
sex = as.numeric(c(1, 2, 1, 2))
father = as.character(c(NA, NA, 11, 11))
mother = as.character(c(NA, NA, 12, 12))
pedinfo = data.frame(pedigree, member, sex, father, mother, stringsAsFactors = FALSE)
set.seed(1)
inher = sim.recomb(pedinfo, 3500) # on a hypothetical chromosome

nsnp = 100000
marker = seq(0,3500,length.out=nsnp)
freq = runif(nsnp, 0.05, 0.95)
haplo = sim.haplotype(freq, 4)
geno = populate.snp(inher, haplo, marker, output.allele = FALSE)

# simulation truth
ibd.proportion(inher,3,4)

# different GRM estimates
grm.pair(geno[3,], geno[4,], freq, method = "twostep")
grm.pair(geno[3,], geno[4,], freq, method = "classic")
grm.pair(geno[3,], geno[4,], freq, method = "robust")
grm.pair(geno[3,], geno[4,], freq, method = "general", weights = sample(freq, nsnp)/sum(freq))

# compute the relatedness matrix
grm.matrix(geno, freq)
grm.matrix(geno, freq, method = "robust")
**ibd.length**  

**Score IBD length.**

**Description**

ibd.length returns the total length of IBD segment between two haplotypes.

**Usage**

ibd.length(inher.hap1, inher.hap2, startpos = NULL, endpos = NULL)

**Arguments**

inher.hap1, inher.hap2
numeric matrix.

startpos, endpos
non-negative number.

**Details**

This function works with output from sim.recomb.

**Value**

A non-negative number representing the length of IBD segment in Haldane centiMorgan.

**Examples**

# a simple pedigree with sibling marriage
pedigree = as.character(rep(1, 5))
member = as.character(c(11, 12, 21, 22, 31))
sex = as.numeric(c(1, 2, 1, 2, 1))
father = as.character(c(NA, NA, 11, 11, 21))
mother = as.character(c(NA, NA, 12, 12, 22))
pedinfo = data.frame(pedigree, member, sex, father, mother, stringsAsFactors = FALSE)
inheritance = sim.recomb(pedinfo, 100)

# IBD length between the two haplotypes of inbred individual 31
ibd.length(inheritance[[9]], inheritance[[10]])
ibd.marker  

Score IBD sharing at a list of marker positions.

Description

ibd.marker determines pairwise IBD sharing at marker positions.

Usage

ibd.marker(inheritance, marker, ind1index, ind2index = NULL, relatedness = TRUE)

Arguments

inheritance  list of numeric matrices.
marker numeric vector.
ind1index, ind2index positive integer, represents index of individual in pedigree.
relatedness logical, determines coding of IBD information.

Details

When only index of one individual is supplied, IBD sharing status at each marker is coded as 0 (not IBD) or 1 (IBD) between the two haplotypes of the individual.

When indices of two individuals are supplied, IBD sharing status at each marker is either in relatedness (default) or lexicographical order of IBD state, where recoding can be done using recode.ibd.

Value

A numeric vector of IBD sharing status at the list of marker positions.

Examples

# a simple pedigree with sibling marriage
pedigree = as.character(rep(1, 5))
member = as.character(c(11, 12, 21, 22, 31))
sex = as.numeric(c(1, 2, 1, 2, 1))
father = as.character(c(NA, NA, 11, 11, 21))
mother = as.character(c(NA, NA, 12, 12, 22))
pedinfo = data.frame(pedigree, member, sex, father, mother, stringsAsFactors = FALSE)
inheritance = sim.recomb(pedinfo, 100)
nsnp = 10
marker = sort(runif(nsnsp, 0, 100))

# IBD at markers between the two haplotypes of the inbred individual
ibd.marker(inheritance, marker, 5)

# IBD at markers between the two full sibs, with different IBD coding
**ibd.proportion**

Score IBD proportion.

**Description**

ibd.proportion returns the proportion of IBD sharing between two haplotypes of the same individual or two individuals.

**Usage**

```r
ibd.proportion(inheritance, ind1index, ind2index = NULL, startpos = NULL, endpos = NULL)
```

**Arguments**

- `inheritance`: list of matrices.
- `ind1index`, `ind2index`: positive integer.
- `startpos`, `endpos`: non-negative number.

**Details**

When only one individual index is supplied, ibd.proportion returns the realized inbreeding coefficient of the individual. When two individual indices are supplied, ibd.proportion returns the realized relatedness of the two individuals.

**Value**

A value between 0 and 1 representing the proportion of IBD segment.

**Examples**

```r
# a simple pedigree with sibling marriage
pedigree = as.character(rep(1, 5))
member = as.character(c(11, 12, 21, 22, 31))
sex = as.numeric(c(1, 2, 1, 2, 1))
father = as.character(c(NA, NA, 11, 11, 21))
mother = as.character(c(NA, NA, 12, 12, 22))
pedinfo = data.frame(pedigree, member, sex, father, mother, stringsAsFactors = FALSE)
inheritance = sim.recomb(pedinfo, 100)

# realized inbreeding of inbred child
get.pedindex(pedinfo, "31")
ibd.proportion(inheritance, 5)
```
# realized relatedness between individual 21 and 22 (parents of inbred child)
get.pedindex(pedinfo, c("21", "22"))
ibd.proportion(inheritance, 3, 4)

### Description

`ibd.segment` determines the starting and ending genetic positions of segments with different amount of pairwise IBD sharing.

#### Usage

```r
ibd.segment(inheritance, ind1index, ind2index = NULL, relatedness = TRUE)
```

#### Arguments

- `inheritance`: list of numeric matrices.
- `ind1index, ind2index`: positive integer, represents index of individual in pedigree.
- `relatedness`: logical, determines coding of IBD information.

#### Details

When only index of one individual is supplied, IBD sharing status for each segment is coded as 0 (not IBD) or 1 (IBD) between the two haplotypes of the individual.

When indices of two individuals are supplied, IBD sharing status for each segment is either in relatedness (default) or lexicographical order of IBD state, where recoding can be done using `recode.ibd`.

#### Value

A dataframe of three variables. `ibd` represents IBD sharing status of a segment, and `startpos/endpos` represents starting/ending genetic position of the segment.

#### Examples

```r
# a simple pedigree with sibling marriage
pedigree = as.character(rep(1, 5))
member = as.character(c(11, 12, 21, 22, 31))
sex = as.numeric(c(1, 2, 1, 2, 1))
father = as.character(c(NA, NA, 11, 11, 21))
mother = as.character(c(NA, NA, 12, 12, 22))
pedinfo = data.frame(pedigree, member, sex, father, mother, stringsAsFactors = FALSE)
inheritance = sim.recomb(pedinfo, 100)

# IBD segments between the two haplotypes of the inbred individual
```
ibd.segment(inheritance, 5)

# IBD segments between the two full sibs
ibd.segment(inheritance, 3, 4) # relatedness
ibd.segment(inheritance, 3, 4, relatedness = FALSE) # lexicographical order of IBD state

---

**ld.weights**

**LD weights**

Description

`ld.weights` computes LD weights for all markers, which is subsequently used to compute LD weighted GRM.

Usage

`ld.weights(data, input.genotype = TRUE)`

Arguments

- `data` numeric matrix.
- `input.genotype` logical.

Details

data can either be the subject by marker numeric genotype matrix (with 0, 1 or 2 coding), or the matrix of marker genotypic correlations. The default option is to input genotype matrix.

Value

A numeric vector of weights. Note that the sum of weights is not constrained to be 1. They should be scaled appropriately before computing the LD weighted GRM.

Examples

# simulate genotypes of 500 individuals at 100 markers
nsnp = 100 # number of SNPs
freq = runif(nsnp, 0.05, 0.95)
nhaplo = 1000 # number of founder haplotypes
haplo.mat = sim.haplotype(freq, nhaplo)
geno.mat = t(sapply(c(1:500), function(x) 4 - haplo.mat[2*x-1,] - haplo.mat[2*x,]))

# compute unconstrained LD weights
ld.weights(geno.mat)
**Description**

`populate.snp` assigns alleles to markers, given inheritance information and founder haplotypes.

**Usage**

```r
populate.snp(inheritance, haplotype, marker, member.index = NULL,
             output.allele = TRUE, output.haplotype = FALSE)
```

**Arguments**

- `inheritance`: list of numeric matrices.
- `haplotype`: numeric matrix.
- `marker`: numeric vector.
- `member.index`: integer vector.
- `output.allele`: logical.
- `output.haplotype`: logical.

**Details**

`inheritance` is a list of matrices produced by, e.g., `sim.recomb`. Each matrix contains a column of founder genome labels and a column of recombination breakpoints for the corresponding meiosis. `haplotype` is a numeric matrix. The matrix is number of haplotypes by number of markers in dimension. Standard coding in this package is 1 for reference allele and 2 for alternate allele. This coding is required when `output.allele = FALSE`. Input data with different coding of alleles can be recoded using `recode.snpdata`. Number of haplotypes cannot be fewer than the number of founder genome labels in `inheritance`. The haplotypes will be assigned to each founder genome label in given order. `marker` is a vector of marker genetic positions in Haldane centiMorgan in ascending order. Range of marker positions cannot exceed range covered by `inheritance`. `member.index` contains indices of members in the pedigree that we wish to output data. Default value is `FALSE`, in which case marker data on everyone will be produced. `get.pedindex` can help find indices given member ID. `output.allele` determines if one or two numbers will be used to represent data at each marker. Default is `TRUE`, in which case marker data is represented by two ordered (paternal first) alleles. Otherwise marker data is represented by a single number (0, 1 or 2) of reference alleles. `output.haplotype` determines if haplotype data are separate in output. It is only used when `output.allele = TRUE`. Default value is `FALSE`, in which case each row in the output matrix represents ordered genotypes from all markers of the same individual. Otherwise each row in the output matrix represents a parental haplotype.
Value

A matrix of genotypic/haplotypic data. The matrix is in individual major, where marker data for each individual/meiosis are found on the same row. Exact format of the matrix depends on various input arguments.

Examples

```r
# a simple pedigree with sibling marriage
pedigree = as.character(rep(1, 5))
member = as.character(c(11, 12, 21, 22, 31))
sex = as.numeric(c(1, 2, 1, 2, 1))
father = as.character(c(NA, NA, 11, 11, 21))
mother = as.character(c(NA, NA, 12, 12, 22))
pedinfo = data.frame(pedigree, member, sex, father, mother, stringsAsFactors = FALSE)

L = 100.0 # segment length
nsnp = 10 # number of SNPs
nhaplo = 4 # number of founder haplotypes
inher = sim.recomb(pedinfo, L)
haplo = matrix(c(3,4,4,4), nhaplo, nsnp)
marker = sort(runif(nsnp, 0, L))

# output genotype data for the 4th and 5th member
# of pedigree, genotype data displayed as two alleles
populate.snp(inher, haplo, marker, c(4, 5))

# output haplotype data for the 4th and 5th member of pedigree
populate.snp(inher, haplo, marker, c(4, 5), output.haplotype = TRUE)

# output genotype data for all members, genotype data
# displayed as counts of reference alleles
gen = recode.snpdata(haplo, input.haplotype = TRUE, output.haplotype = TRUE)[[1]]
populate.snp(inher, gen, marker, output.allele = FALSE)
```

read.plink.binary  
Read PLINK binary file.

Description

read.plink.binary reads PLINK binary .bed file and the corresponding .bim and .fam file.

Usage

```r
read.plink.binary(bed, bim = NULL, fam = NULL, na.strings = c("0", "-9"))
```

Arguments

- `bed, bim, fam`  
  PLINK files with appropriate extensions.
- `na.strings`  
  string vector, text entries to be treated as NA's.
Details

When the three files have the same name, only the .bed file needs to be specified.

Value

A list of three elements: genotype, fam and map. To be consistent with PLINK .bed file, genotype is a n_subject by n_marker matrix of counts of reference alleles. Missing values are -9. fam is a dataframe that contains the first six columns of a PLINK .ped file. map is a dataframe that contains the four columns of a PLINK .map file, with two additional columns: allele_1 for the reference allele type, allele_2 for the alternate allele type.

read.plink.text

Read PLINK text file.

Description

read.plink.text reads PLINK text files in either the original or transposed format.

Usage

read.plink.text(ped, map = NULL, output.allele = TRUE, na.strings = c("0", "-9"))

Arguments

ped, map PLINK files with appropriate extensions.
output.allele logical, default is to output genotype as alleles.
na.strings Character vector, set of characters to be treated as missing values.

Details

The PLINK pedigree file should be supplied with the appropriate extension. The corresponding map file can be omitted if it has the same file name as the pedigree file and has the appropriate extension.

Value

A list of three elements: genotype, fam and map. To be consistent with PLINK .bed file, genotype by default is a n_subject by (2 x n_marker) matrix of alleles, where 1 represents the reference allele and 2 the alternate allele. Alternatively, genotype can be outputted as 0, 1 or 2 copies of reference allele count by using output.allele = FALSE. Missing values are -9. fam is a dataframe that contains the first six columns of a PLINK .ped file. map is a dataframe that contains the four columns of a PLINK .map file, with two additional columns: allele_1 for the reference allele type, allele_2 for the alternate allele type.
**recode.ibd**

*Recode IBD sharing.*

**Description**

recode.ibd recodes pairwise IBD sharing information.

**Usage**

recode.ibd(ibdvec, from, to)

**Arguments**

ibdvec numeric vector of input IBD sharing information.

from, to string, IBD sharing information options include "ibdstate", "lexi", "jac", "jac.red" and "relatedness".

**Details**

At any marker, there are 15 possible IBD states between the four genes of two individuals. "ibd-state" represents the standard coding of the 15 states from 1111 to 1234. "lexi" and "jac" represent lexicographical and Jacquard ordering of "ibdstate" from 1 to 15 respectively. "jac.red" is a condensed Jacquard ordering from 1 to 9 for the genotypically distinct groups of IBD states when phasing is unknown. "relatedness" refers to local relatedness coefficient taking values in (0, 0.5, 1, 2).

"ibdstate", "lexi" and "jac" are of the highest level (complete information), "jac.red" is of mid level, whereas "relatedness" is of the lowest level. Conversion cannot go from lower level to higher level.

**Value**

A numeric vector of recoded IBD states.

**Examples**

test.state = c(1111, 1122, 1212, 1222, 1234)
recode.ibd(test.state, "ibdstate", "lexi")
recode.ibd(test.state, "ibdstate", "jac")
recode.ibd(test.state, "ibdstate", "jac.red")
recode.ibd(test.state, "ibdstate", "relatedness")
recode.snpdata  

Recode SNP marker data.

Description

recode.snpdata recodes SNP marker data for use with other functions in this package.

Usage

recode.snpdata(data, snp.major = FALSE, ma.ref = FALSE,  
input.haplotype = FALSE, output.allele = TRUE, output.haplotype = FALSE,  
na.string = NULL)

Arguments

data  
numeric matrix or dataframe.
snp.major  
logical.
ma.ref  
logical.
input.haplotype  
logical.
output.allele  
logical.
output.haplotype  
logical.
na.string  
numeric or character vector.

Details

The standard marker data used by other functions of this package takes one of three forms: (a) subjects by row, counts of reference alleles by column; (b) subjects by row, allelic types (2 per marker) by column; (c) haplotypes (2 per subject) by row, allelic types by column. Reference alleles are coded 1, alternate alleles are coded 2.

By default, snp.major = FALSE, set it to TRUE if input matrix has SNPs by row and allelic types (2 per subject) by column. ma.ref = FALSE, set it to TRUE if the minor allele is to be the reference allele. input.haplotype = FALSE, set it to TRUE if input matrix has haplotypes (2 per subject) by row and allelic types by column. output.allele = TRUE, set it to FALSE if counts of reference alleles is the desired output format. output.haplotype = FALSE, set it to TRUE if recoded marker data by haplotype is the desired output format. input.haplotype is only invoked when snp.major = FALSE. output.haplotype is only invoked when output.allele = TRUE.

Value

A list of two elements. First element named data is a matrix of recoded marker data in specified format. Second element is a dataframe named alleles that specifies reference/alternate alleles at all markers.
Examples

test.dat = matrix(c(3,4,4,3), 4, 10)

# treat test.dat as 4 input haplotypes of two subjects at 10 SNP markers,
# output recoded data as haplotypes
recode.snpdata(test.dat, input.haplotype = TRUE, output.haplotype = TRUE)

# treat test.dat as 4 input haplotypes of two subjects at 10 SNP markers,
# output recoded data as counts of reference alleles
recode.snpdata(test.dat, input.haplotype = TRUE, output.allele = FALSE)

# treat test.dat as allelic types at 5 SNPs of 4 subjects,
# output recoded data as haplotypes
recode.snpdata(test.dat, output.haplotype = TRUE)

---

sim.haplotype

Simulate artificial haplotypes.

Description

sim.haplotype returns haplotypes of the specified number of SNPs simulated under linkage equilibrium.

Usage

sim.haplotype(freq, nhaplo)

Arguments

freqvector of values between 0 and 1.

nhaplo positive integer.

Details

freq are reference allele frequencies. nhaplo haplotypes are simulated independently.

Value

A matrix of nhaplo rows and length(freq) columns. Reference alleles are coded 1, alternate alleles are coded 2.

Examples

nsnp = 7 # number of SNPs
dfreq = runif(nsnip, 0.05, 0.95)
nhaplo = 4 # number of founder haplotypes
sim.haplotype(freq, nhaplo)
sim.recomb

Simulate inheritance on a given pedigree.

Description

sim.recomb returns inheritance information simulated on a given pedigree over the specified segment length.

Usage

sim.recomb(pedinfo, seglength)

Arguments

- pedinfo: dataframe.
- seglength: positive real number.

Details

pedinfo must contain at least the following components: unique individual ID named member, father and mother ID named father and mother, and sex (1 for male, 2 for female) named sex. Parents must precede offsprings. Pedigree founders are treated as unrelated.

seglength represents length of genomic segment in Haldane centiMorgan. Recombination breakpoints are simulated under a homogeneous Poisson process with rate seglength/100.

Value

A list of matrices for each meiosis. Each matrix has two columns: founder genome labels (fgl) and recombination breakpoints (recomb). Paternal meiosis precedes maternal meiosis.

Examples

# a simple pedigree with sibling marriage
pedigree = as.character(rep(1, 5))
member = as.character(c(11, 12, 21, 22, 31))
sex = as.numeric(c(1, 2, 1, 2, 1))
father = as.character(c(NA, NA, 11, 11, 21))
mother = as.character(c(NA, NA, 12, 12, 22))
pedinfo = data.frame(pedigree, member, sex, father, mother, stringsAsFactors = FALSE)

# simulate inheritance over a segment of 100 centiMorgan
sim.recomb(pedinfo, 100)
**write.ibdhaplo**  
*Write IBDHAPLO*

---

**Description**

`write.ibdhaplo` prepares the marker data file for running IBDHAPLO.

**Usage**

```r
write.ibdhaplo(marker, freq, data, member, input.allele = TRUE,  
               input.haplotype = FALSE, outfile = tempfile("ibdhaplo", fileext = ".txt"))
```

**Arguments**

- `marker`: numeric vector, marker genetic positions in cM.
- `freq`: numeric vector, marker reference allele frequencies.
- `data`: numeric matrix, genetic marker data.
- `member`: string vector, member ID.
- `input.allele`: logical, default TRUE.
- `input.haplotype`: logical, default FALSE.
- `outfile`: string, output file name.

**Details**

The input marker data needs to be subject/haplotype by marker/allele. For example, suppose `data` is a 4x10 matrix, use `input.allele = FALSE` if `data` contains counts of reference alleles of 4 individuals at 10 markers; use `input.haplotype = TRUE` if `data` contains allelic types of 4 haplotypes at 10 markers; use default options if `data` contains allelic types of 4 individuals at 5 markers.

**References**


**Examples**

```r
## Not run:
nsnp = 7 # number of SNPs
freq = runif(nsnp, 0.05, 0.95)
nhaplo = 4 # number of founder haplotypes
haplotype = sim.haplotype(freq, nhaplo)
marker = sort(runif(7,0,100))
write.ibdhaplo(marker, freq, haplotype, member = c("ind1", "ind2"),
               input.haplotype = TRUE)

## End(Not run)
```
Index

check.pedinfo, 2
fgl2ibd, 2
fgl2relatedness, 3

get.pedindex, 4, 12
grm.matrix, 4, 6
grm.pair, 5, 5

ibd.length, 7
ibd.marker, 8
ibd.proportion, 9
ibd.segment, 10

ld.weights, 11

populate.snp, 12

read.plink.binary, 13
read.plink.text, 14
recode.ibd, 3, 8, 10, 15
recode.snpdata, 12, 16

sim.haplotype, 17
sim.recomb, 7, 12, 18

write.ibdhaplo, 19