Package 'GHap'

February 18, 2017

Type Package

Title Genome-Wide Haplotyping

Version 1.2.2

Date 2017-02-18

Author Yuri Tani Utsunomiya, Marco Milanesi

Maintainer Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Description Haplotype calling from phased SNP data.

License GPL (>= 2)

Depends bigmemory (>= 4.4.6), parallel (>= 3.0.0), Matrix (>= 1.2-6), methods (>= 3.3.0), lme4 (>= 1.1-12)

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation no

Repository CRAN

Date/Publication 2017-02-18 21:22:55

R topics documented:

`ghap.ancestral` *Inference of haplotype origin*

Description

Given one test population and two parental populations, the function assigns haplotype alleles in the test population to one of the parental populations.

Usage

```
ghap.ancestral(hapstats.test, hapstats.parent1, hapstats.parent2, freq = 0.05,
 prob.assign = 0.60)
```

Arguments

`hapstats.test`	A data.frame containing haplotype statistics for the population to be tested, as generated by the `ghap.hapstats` function.
`hapstats.parent1`	A data.frame containing haplotype statistics for the first parental population.
`hapstats.parent2`	A data.frame containing haplotype statistics for the second parental population.
`freq`	The haplotype frequency threshold used to filter haplotype alleles in the parental populations (default = 0.05). For each parental population, if the allele frequency is lower than the threshold the probability of origin is automatically set to zero.
`prob.assign`	The probability threshold used for the assignment test (default = 0.60). Haplotypes that are not assigned to any of the parental populations are also marked as unassigned (UNK).

Details

This function calculates the probability that one haplotype from a tested population was inherited from one of the tested parental populations. The function followed the method described by Bolormaa et al. (2011).

Value

The function returns a dataframe and a file with the following columns:

BLOCK	Block alias.
CHR	Chromosome name.
BP1	Block start position.
BP2	Block end position.
ALLELE	Haplotype allele identity.
FREQ.TEST	Haplotype frequency in the test population.
FREQ.PARENT[1 and 2]	Haplotype frequency in the first and second parental populations, respectively.
PROB.PARENT[1 and 2]	Assignment probabilities calculated following Bolormaa et al. (2011).
ORIGIN	Haplotype origin (PARENT1, PARENT2 or UNK).

Author(s)

Marco Milanesi <marco.milanesi.mm@gmail.com>

References

S. Bolormaa et al. Detection of chromosome segments of zebu and taurine origin and their effect on beef production and growth. J. Anim. Sci. 2011. 89:2050-2060.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
```

```
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# # Compute haplotype allele statistics for each group
# haplo <- ghap.subsethaplo(haplo,haplo$id,rep(TRUE,times=haplo$nalleles))
# ASW.ids <- unique(haplo$id[haplo$pop=="ASW"])
# YRI.ids <- unique(haplo$id[haplo$pop=="YRI"])
# CEU.ids <- unique(haplo$id[haplo$pop=="CEU"])
# haplo <- ghap.subsethaplo(haplo,YRI.ids,haplo$allele.in)
# YRI.hapstats <- ghap.hapstats(haplo,ncores = 2)
# haplo <- ghap.subsethaplo(haplo,CEU.ids,haplo$allele.in)
# CEU.hapstats <- ghap.hapstats(haplo,ncores = 2)
# haplo <- ghap.subsethaplo(haplo,ASW.ids,haplo$allele.in)
# ASW.hapstats <- ghap.hapstats(haplo,ncores = 2)
# haplo <- ghap.subsethaplo(haplo,haplo$id,rep(TRUE,times=haplo$nalleles))
#
# # Find haplotype origin
# # ASW is the test population. YRI and CEU are used as parental populations
# # The frequency threshold is set to 0.05 and the probability of assignment to 0.60
# ancestry <- ghap.ancestral(ASW.hapstats, YRI.hapstats, CEU.hapstats, 0.05, 0.60)
# ancestry <- ancestry[ancestry$FREQ.TEST > 0,]
```

`ghap.assoc` *Association analysis for HapAlleles*

Description

Given a GHap.lmm object (as supplied by the `ghap.lmm` function), this function returns association statistics for HapAlleles.

Usage

```
ghap.assoc(response, haplo, weights=NULL, gc = TRUE,
           only.active.alleles = TRUE, ncores = 1)
```

Arguments

`response`	A vector of phenotypes. The vector must be named and all names must be present in the GHap.haplo object.
`weights`	A numeric vector with weights for phenotypes. These weights are treated as diagonal elements of the inverse weight matrix. If not supplied, the analysis is carried out assuming all observations are equally important.

haplo	A GHap.haplo object.
only.active.alleles	A logical value specifying whether calculations should be reported only for active haplotype alleles (default = TRUE).
gc	A logical value specifying whether genomic control should be performed (default = TRUE). Currently, this option does not take effect on HapBlocks.
ncores	A numeric value specifying the number of cores to be used in parallel computations (default = 1).

Details

This function uses least squares regression to test each HapAllele at a time for association with phenotypes. The fixed effect, error variance and test statistic of a given HapAllele are estimated as:

$$\hat{a}_i = (\mathbf{x}_i'\mathbf{x}_i)^{-1}\mathbf{x}_i'\mathbf{y}$$

$$VAR(\hat{a}_i) = (\mathbf{x}_i'\mathbf{x}_i)^{-1}\hat{\sigma}_e^2$$

$$t_i^2 = \frac{\hat{a}_i^2}{VAR(\hat{a}_i)}$$

Under the null hypothesis that the regression coefficient is zero $t_i^2 \sim \chi^2(\nu = 1)$. This function supports repeated measures, and records are dully mapped against the vectors of HapGenotypes.

If the vector of responses comprises adjusted records (i.e., residuals) from a linear mixed model, the regression analysis approximates the model implemented in other GWAS tools. However, the user must be aware of two known caveats associated with this approach. First, by pre-adjusting records instead of estimating HapAllele effects based on generalized least squares equations we ignore covariance structure and therefore bias the estimates downwards (Svishcheva et al., 2012). Second, each HapAllele being tested was also potentially included in the kinship matrix in the mixed model analysis, such that the HapAllele is included twice in the model: as fixed and random effect. This problem is known as proximal contamination (Listgarten et al., 2012). In the first case, we can use genomic control to recover p-values to an unbiased scale (Devlin and Roeder, 1999; Amin et al., 2007). However, not much can be done regarding the estimates of the effects. As a general recommendation, if the user is only interested in p-values, the ghap.assoc analysis should be sufficient. When effect estimates are of interest, the user can include the candidate HapAllele as a fixed effect in the full model in ghap.lmm. For the second case, a leave-one-chromosome-out (LOCO analysis) procedure can mitigate proximal contamination (Yang et al., 2014).

Value

The function returns a data.frame with the following columns:

BLOCK	Block alias.
CHR	Chromosome name.
BP1	Block start position.
BP2	Block end position.
ALLELE	Haplotype allele identity.

BETA	BLUE for the fixed effect of the haplotype allele.
SE	Standard error for the fixed effect.
FREQ	Frequency of the haplotype allele.
CHISQ.OBS	Observed value for the test statistics. If gc = TRUE (default), these values are scaled by the inflation factor. Inflation is computed through regression of observed quantiles onto expected quantiles. In order to avoid overestimation, only HapAlleles with test statistics within three standard deviations from the mean are used to compute the inflation factor.
CHISQ.EXP	Expected values for the test statistics.
logP	log10(1/P) or -log10(P) for the fixed effect.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

N. Amin et al. A Genomic Background Based Method for Association Analysis in Related Individuals. PLoS ONE. 2007. 2:e1274.

Y. Da. Multi-allelic haplotype model based on genetic partition for genomic prediction and variance component estimation using SNP markers. BMC Genet. 2015. 16:144.

B. Devlin and K. Roeder. Genomic control for association studies. Biometrics. 1999. 55:997-1004.

J. Listgarten et al. Improved linear mixed models for genome-wide association studies. Nat. Methods. 2012. 9:525-526.

G. R. Svishcheva et al. Rapid variance components-based method for whole-genome association analysis. Nat Genet. 2012. 44:1166-1170.

J. Yang et al. Advantages and pitfalls in the application of mixed-model association methods. Nat. Genet. 2014. 46: 100-106.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
```

```
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[haplo$pop %in% c("TSI","CEU")]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,rep(TRUE,times=haplo$nalleles))
# hapstats <- ghap.hapstats(haplo, ncores = 2)
# common <- hapstats$TYPE %in% c("REGULAR","MAJOR") &
#  hapstats$FREQ > 0.05 &
#  hapstats$FREQ < 0.95
# haplo <- ghap.subsethaplo(haplo,EUR.ids,common)
#
# #Compute relationship matrix
# K <- ghap.kinship(haplo, batchsize = 100)
#
# # Quantitative trait with 50% heritability
# # Unbalanced repeated measurements (0 to 30)
# # Two major haplotypes accounting for 50% of the genetic variance
# myseed <- 123456789
# set.seed(myseed)
# major <- sample(which(haplo$allele.in == TRUE),size = 2)
# g2 <- runif(n = 2, min = 0, max = 1)
# g2 <- (g2/sum(g2))*0.5
# sim <- ghap.simpheno(haplo, kinship = K, h2 = 0.5, g2 = g2, nrep = 30,
#                      balanced = FALSE, major = major, seed = myseed)
#
# #Fit model using REML
# model <- ghap.lmm(fixed = phenotype ~ 1, random = ~ individual,
#                   covmat = list(individual = K), data = sim$data)
#
#
# ### RUN ###
#
# #HapAllele GWAS using GEBVs as response
# pheno <- model$random$individual
# gwas1 <- ghap.assoc(response = pheno, haplo = haplo, ncores = 4)
#
# #HapAllele GWAS using GEBVs as response
# #Weight observations by number of repeated measurements
# pheno <- model$random$individual
# w <- table(sim$data$individual)
# w <- w + mean(w)
# w <- w[names(pheno)]
# gwas2 <- ghap.assoc(response = pheno, haplo = haplo, ncores = 4, weights = w)
#
# #HapAllele GWAS using residuals as response
# pheno <- model$residuals
```

```
# names(pheno) <- sim$data$individual
# gwas3 <- ghap.assoc(response = pheno, haplo = haplo, ncores = 4)
#
# #Plot results
# plot(gwas1$BP1/1e+6,gwas1$logP,pch=20,col="darkgreen",ylim=c(0,20),
#      xlab="Position (in Mb)",ylab=expression(-log[10](p)))
# points(gwas2$BP1/1e+6,gwas2$logP,pch=20,col="gray")
# points(gwas3$BP1/1e+6,gwas3$logP,pch=20,col="blue")
# abline(v=haplo$bp1[major]/1e+6,lty=3)
# abline(h=-log10(0.05/nrow(gwas1)),lty=3)
# legend("topleft",legend = c("GEBVs","weighted GEBVs","residuals"),
#        pch = 20,col=c("darkgreen","gray","blue"))
```

`ghap.blockgen` *Haplotype block generator*

Description

This function generates HapBlocks based on sliding windows. The window and the step size can be specified in markers or kbp. For each window, block coordinates are generated.

Usage

```
ghap.blockgen(phase, windowsize = 10, slide = 5, unit = "marker", nsnp = 2)
```

Arguments

`phase`	A GHap.phase object
`windowsize`	A numeric value for the size of the window (default = 10).
`slide`	A numeric value for the step size (default = 5).
`unit`	A character value for the size unit used for the window and the step. It can be either "marker" or "kbp" (default = "marker").
`nsnp`	A numeric value for the minimum number of markers per block.

Value

A data frame with columns:

`BLOCK`	Block alias.
`CHR`	Chromosome name.
`BP1`	Block start position.
`BP2`	Block end position.
`SIZE`	Haplotype size.
`NSNP`	Number of marker.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# ### RUN ###
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate blocks of 100 kb sliding 100 kbp at a time
# blocks.kb <- ghap.blockgen(phase, windowsize = 100, slide = 100, unit = "kbp")
```

`ghap.blockstats` *Block statistics*

Description

Generate block summary statistics from pre-computed haplotype statistics.

Usage

```
ghap.blockstats(hapstats, ncores=1)
```

Arguments

`hapstats`	A data.frame containing haplotype statistics, as generated by the `ghap.hapstats` function.
`ncores`	A numeric value specifying the number of processors to be used in parallelization of independent Markov chains (default = 1).

Details

For each haplotype block, the function counts the number of unique haplotype alleles and computes the expected heterozygosity $1 - \sum {p_i}^2$, where p_i is the frequency of HapAllele *i*. Please notice that when haplotypes are prunned out by frequency the block statistics can retrieve high expected heterozygosity for blocks with small number of HapAlleles.

Value

A data frame with columns:

BLOCK	Block alias.
CHR	Chromosome name.
BP1	Block start position.
BP2	Block end position.
EXP.H	Block expected heterozygosity.
N.ALLELES	Number of HapAlleles per block.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
# #Compute haplotype statistics
# hapstats <- ghap.hapstats(haplo, ncores = 2)
#
#
```

```
# ### RUN ###
#
# #Compute block statistics
# blockstats <- ghap.blockstats(hapstats, ncores = 2)
```

---

ghap.blup *Convert breeding values into BLUP solutions of HapAllele effects*

Description

Given genomic estimated breeding values (GEBVs), compute Best Linear Unbiased Predictor (BLUP) solutions for HapAllele effects.

Usage

```
ghap.blup(gebvs,haplo,invcov, gebvsweights = NULL, haploweights = NULL, nperm = 1,
          only.active.alleles = TRUE, ncores = 1)
```

Arguments

gebvs	A vector of GEBVs. The vector must be named and all names must be present in the GHap.haplo object.
haplo	A GHap.haplo object.
invcov	The inverse covariance (i.e., genomic kinship) matrix for GEBVs.
gebvsweights	A numeric vector providing individual-specific weights.
haploweights	A numeric vector providing HapAllele-specific weights.
nperm	Number of permutations to be performed for significance assessment (default = 1).
only.active.alleles	A logical value specifying whether only active haplotype alleles should be included in the calculations (default = TRUE).
ncores	A numeric value specifying the number of cores to be used in parallel computing (default = 1).

Details

The function uses the equation:

$$\hat{\mathbf{a}} = q\mathbf{D}\mathbf{M}'\mathbf{K}^{-1}\hat{\mathbf{u}}$$

where $\mathbf{M}$ is the *N* x *H* centered matrix of HapGenotypes observed for *N* individuals and *H* HapAlleles, $\mathbf{D} = diag(d_i)$, d_i is the weight of HapAllele *i* (default $d_i = 1$), q is the inverse weighted sum of variances in the columns of $\mathbf{M}$, $\mathbf{K}$ is the haplotype-based kinship matrix and $\hat{u}$ is the vector of GEBVs. The permutation procedure consists in randomizing the vector $\hat{\mathbf{u}}$ and computing the null statistic $max(\hat{\mathbf{a}})$. The permutation p-value is computed as the number of times the HapAllele effect was smaller than the null statistic.

Value

The function returns a dataframe with columns:

BLOCK	Block alias.
CHR	Chromosome name.
BP1	Block start position.
BP2	Block end position.
ALLELE	Haplotype allele identity.
SCORE	BLUP for the random effect of the haplotype allele.
FREQ	Frequency of the haplotype allele.
VAR	Variance in allele-specific breeding values.
pVAR	Proportion of variance explained by the haplotype allele.
CENTER	Average genotype (meaningful only for predictions with ghap.profile).
SCALE	A constant set to 1 (meaningful only for predictions with ghap.profile).
P	P-value for the permutation test. This column is suppressed if nperm < 1.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

I. Stranden and D.J. Garrick. Technical note: derivation of equivalent computing algorithms for genomic predictions and reliabilities of animal merit. J Dairy Sci. 2009. 92:2971-2975.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
```

```
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[haplo$pop %in% c("TSI","CEU")]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,rep(TRUE,times=haplo$nalleles))
# hapstats <- ghap.hapstats(haplo, ncores = 2)
# common <- hapstats$TYPE %in% c("REGULAR","MAJOR") &
#  hapstats$FREQ > 0.05 &
#  hapstats$FREQ < 0.95
# haplo <- ghap.subsethaplo(haplo,EUR.ids,common)
#
# #Compute relationship matrix
# K <- ghap.kinship(haplo, batchsize = 100)
#
# # Quantitative trait with 50% heritability
# # Unbalanced repeated measurements (0 to 30)
# # Two major haplotypes accounting for 50% of the genetic variance
# myseed <- 123456789
# set.seed(myseed)
# major <- sample(which(haplo$allele.in == TRUE),size = 2)
# g2 <- runif(n = 2, min = 0, max = 1)
# g2 <- (g2/sum(g2))*0.5
# sim <- ghap.simpheno(haplo, kinship = K, h2 = 0.5, g2 = g2, nrep = 30,
#                      balanced = FALSE, major = major, seed = myseed)
#
# #Fit model using REML
# model <- ghap.lmm(fixed = phenotype ~ 1, random = ~ individual,
#                   covmat = list(individual = K), data = sim$data)
#
#
# ### RUN ###
#
# #BLUP GWAS
# gebvs <- model$random$individual
# gebvsw <- table(sim$data$individual)
# gebvsw <- gebvsw + mean(gebvsw)
# gebvsw <- gebvsw[names(gebvs)]
# Kinv <- ghap.kinv(K)
# gwas.blup <- ghap.blup(gebvs = gebvs, haplo = haplo, gebvsweights = gebvsw,
#                        ncores = 4, invcov = Kinv)
# plot(gwas.blup$BP1/1e+6,gwas.blup$pVAR*100,pch=20,
#      xlab="Position (in Mb)",ylab="Variance explained (%)")
# abline(v=haplo$bp1[major]/1e+6)
# #BLUP with one update
# w <- gwas.blup$VAR*nrow(gwas.blup)
# K2 <- ghap.kinship(haplo=haplo,weights = w)
# Kinv2 <- ghap.kinv(K2)
# gwas.blup2 <- ghap.blup(gebvs = gebvs, haplo = haplo, invcov = Kinv2, ncores = 2,
#                         gebvsweights = gebvsw, haploweights = w)
# plot(gwas.blup2$BP1/1e+6,gwas.blup2$pVAR*100,pch=20,
```

```
#      xlab="Position (in Mb)",ylab="Variance explained (%)")
# abline(v=haplo$bp1[major]/1e+6)
```

---

ghap.fst *Haplotype Fst*

Description

Multi-alleleic Fst computed using block summary statistics generated from ghap.blockstats.

Usage

```
ghap.fst(blockstats.pop1, blockstats.pop2, blockstats.tot)
```

Arguments

blockstats.pop1	A data.frame containing block statistics computed on population 1.
blockstats.pop2	A data.frame containing block statistics computed on population 2.
blockstats.tot	A data.frame containing block statistics computed on population 1 + population 2.

Details

This function calculates Fst (Nei, 1973) based on the formula for multi-allelic markers:

$$Fst = (Ht - Hs)/Ht$$

where *Ht* is the total gene diversity (i.e., expected heterozygosity in the population) and *Hs* is the subpopulation gene diversity (i.e., the average expected heterozygosity in the subpopulations).

Value

The function returns a data.frame with the following columns:

BLOCK	Block alias.
CHR	Chromosome name.
BP1	Block start position.
BP2	Block end position.
EXP.H.pop1	Expected heterozygosity in population 1.
EXP.H.pop2	Expected heterozygosity in population 2.
EXP.H.tot	Expected heterozygosity in the total population.
FST	Fst value.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Marco Milanesi <marco.milanesi.mm@gmail.com>

References

M. Nei. Analysis of Gene Diversity in Subdivided Populations. PNAS. 1973. 70, 3321-3323.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# # Compute haplotype allele statistics for each group
# CHB.ids <- haplo$id[which(haplo$pop=="CHB")]
# CEU.ids <- haplo$id[which(haplo$pop=="CEU")]
# haplo <- ghap.subsethaplo(haplo,CHB.ids,haplo$allele.in)
# CHB.hapstats <- ghap.hapstats(haplo,ncores = 2)
# haplo <- ghap.subsethaplo(haplo,CEU.ids,haplo$allele.in)
# CEU.hapstats <- ghap.hapstats(haplo,ncores = 2)
# haplo <- ghap.subsethaplo(haplo,c(CHB.ids,CEU.ids),haplo$allele.in)
# TOT.hapstats <- ghap.hapstats(haplo,ncores = 2)
# haplo <- ghap.subsethaplo(haplo,haplo$id,rep(TRUE,times=haplo$nalleles))
#
# # Compute haplotype block statistics for each group
# CHB.blockstats <- ghap.blockstats(CHB.hapstats, ncores = 2)
# CEU.blockstats <- ghap.blockstats(CEU.hapstats, ncores = 2)
# TOT.blockstats <- ghap.blockstats(TOT.hapstats, ncores = 2)
#
```

```
# # Calculate Fst
# fst<-ghap.fst(CHB.blockstats, CEU.blockstats, TOT.blockstats)
#
# # Plot results
# top.fst <- fst[fst$FST == max(fst$FST, na.rm=TRUE),]
# plot(
#   x = (fst$BP1+fst$BP2)/2e+6,
#   y = fst$FST, pch = "",
#   ylab = expression(paste("Haplotype ", F[ST])),
#   xlab = "Chromosome 2 (in Mb)",
#   ylim=c(0,1)
# )
# abline(v=108.7, col="gray")
# points(x = (fst$BP1+fst$BP2)/2e+6, y = fst$FST, pch = 20, col="#471FAA99")
# points(x = (top.fst$BP1+top.fst$BP2)/2e+6, y = top.fst$FST, pch = 20, col="red")
# text(x = 125, y = max(fst$FST, na.rm=TRUE), "EDAR", col="red")
# CEU.hapstats[CEU.hapstats$BLOCK == top.fst$BLOCK & CEU.hapstats$FREQ > 0,1:9]
# CHB.hapstats[CHB.hapstats$BLOCK == top.fst$BLOCK & CHB.hapstats$FREQ > 0,1:9]
```

---

ghap.hap2tped *Convert haplotype allele counts to PLINK tped*

Description

This function takes a haplotype genotypes matrix (as generated with the ghap.haplotyping function) and converts it to PLINK tped format.

Usage

```
ghap.hap2tped(infile,batchsize = 500, outfile,verbose = TRUE)
```

Arguments

infile	The prefix for the *.hapsamples*, *.hapalleles* and *.hapgenotypes* files generated by ghap.haplotyping.
batchsize	A numeric value controlling the number of haplotype alleles to be processed at a time (default = 500).
outfile	A character value specifying the name used for the .tped, .tfam and .tref output files.
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

The returned file mimics a standard PLINK (Purcell et al., 2007; Chang et al., 2015) tped file, where haplotype allele counts 0, 1 and 2 are recoded as NN, NH and HH genotypes (N = NULL and H = haplotype allele), as if haplotypes were bi-alelic markers. This codification is acceptable for any given analysis relying on SNP genotype counts, as long as the user specifies that the analysis should be done using the H allele as reference for counts. You can specify reference alleles using the .tref file in PLINK with the *reference-allele* command. This is desired for very large datasets, as softwares such as PLINK and GCTA (Yang et al., 2011) have faster implementations for regression, principal components and kinship matrix analyses. The name for each pseudo-marker is composed by a concatenation (separated by "_") of block name, start, end, and haplotype allele identity. Pseudo-marker positions are computed as (start+end)/2.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Marco Milanesi <marco.milanesi.mm@gmail.com>

References

C. C. Chang et al. Second-generation PLINK: rising to the challenge of larger and richer datasets. Gigascience. 2015. 4, 7.

S. Purcell et al. PLINK: a tool set for whole-genome association and population-based linkage analyses. Am. J. Hum. Genet. 2007. 81, 559-575.

J. Yang et al. GCTA: A tool for genome-wide complex trait analysis. Am. J. Hum. Genet. 2011. 88, 76-82.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
```

```
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# # Subset common haplotypes
# hapstats <- ghap.hapstats(haplo, ncores = 2)
# common <- hapstats$TYPE %in% c("REGULAR","MAJOR") &
#  hapstats$FREQ > 0.05 &
#  hapstats$FREQ < 0.95
# haplo <- ghap.subsethaplo(haplo,unique(haplo$id),common)
#
# # Output GHap.haplo object
# ghap.outhaplo(haplo = haplo, outfile = "humansub")
#
# # Convert to tped
# ghap.hap2tped(infile = "humansub", outfile = "humansub")
```

`ghap.haplotyping` *Haplotype genotypes*

Description

Generate matrix of HapGenotypes for user-defined blocks.

Usage

```
ghap.haplotyping(phase, blocks, outfile, freq = c(0,1), drop.minor = FALSE,
 batchsize = 500, ncores = 1, verbose = TRUE)
```

Arguments

`phase`	A GHap.phase object.
`blocks`	A data frame containing block boundaries, such as supplied by the `ghap.blockgen` function.
`outfile`	A character value specifying the name for the output files.
`freq`	A numeric vector of length 2 specifying the range of haplotype allele frequency to be included in the output. Default is c(0,1), which includes all alleles.
`drop.minor`	A logical value specfying whether the minor allele should be excluded from the output (default = FALSE).
`batchsize`	A numeric value controlling the number of haplotype blocks to be processed and written to output at a time (default = 500).
`ncores`	A numeric value specifying the number of cores to be used in parallel computations (default = 1).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The function outputs three files with suffix:

- **.hapsamples**: space-delimited file without header containing two columns: Population and Individual ID.
- **.hapalleles**: space-delimited file without header containing five columns: Block Name, Chromosome, Start and End Position (in bp), and HapAllele.
- **.hapgenotypes**: space-delimited file without header containing the HapGenotype matrix (coded as 0, 1 or 2 copies of the HapAllele). The dimension of the matrix is *m x n*, where *m* is the number of HapAlleles and *n* is the number of individuals.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Marco Milanesi <marco.milanesi.mm@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
#
# ### RUN ###
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
```

`ghap.hapstats` *Haplotype statistics*

Description

HapAllele summary statistics.

Usage

```
ghap.hapstats(haplo, alpha = c(1,1), only.active.samples = TRUE,
 only.active.alleles = TRUE, ncores = 1)
```

Arguments

haplo	A GHap.haplo object.
alpha	A numeric vector of size 2 specifying the shrinkage parameters for the expected-to-observed homozygotes ratio. Default is c(1,1).
only.active.samples	A logical value specifying whether only active samples should be included in the output (default = TRUE).
only.active.alleles	A logical value specifying whether only active haplotype alleles should be included in the output (default = TRUE).
ncores	A numeric value specifying the number of cores to be used in parallel computations (default = 1).

Value

A data frame with columns:

BLOCK	Block alias.
CHR	Chromosome name.
BP1	Block start position.
BP2	Block end position.
ALLELE	Haplotype allele identity.
N	Number of observations for the haplotype.
FREQ	Haplotype frequency.
O.HOM	Observed number of homozygotes.
O.HET	Observed number of heterozygotes.
E.HOM	Expected number of homozygotes.
RATIO	Shrinkage expected-to-observed ratio for the number of homozygotes.
BIN.logP	log10(1/P) or -log10(P) for Hardy-Weinberg equilibrium assuming number of homozygotes follows a Binomial distribution.
POI.logP	log10(1/P) or -log10(P) for Hardy-Weinberg equilibrium assuming number of homozygotes follows a Poisson distribution.
TYPE	Category of the haplotype: "SINGLETON" = unique haplotype of its block; "ABSENT" = the frequency of the allele is 0; "MINOR" = the least frequent haplotype of its block (in the case of ties, only the first haplotype is marked); "MAJOR" = the most frequent hapotype of its block (ties are also resolved by marking the first haplotype); "REGULAR" = the haplotype does not fall in any of the previous categories. Categories "SINGLETON", "MINOR" and "MAJOR" only apply for blocks where frequencies sum to 1.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Marco Milanesi <marco.milanesi.mm@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# #Compute haplotype statistics
# hapstats <- ghap.hapstats(haplo, ncores = 2)
```

---

ghap.kinship *Kinship matrix from haplotypes*

Description

This function computes a HapAllele-based kinship matrix from a GHap.haplo object.

Usage

```
ghap.kinship(haplo, weights, batchsize = 500, only.active.samples = TRUE,
 only.active.alleles = TRUE, verbose = TRUE)
```

Arguments

haplo	A GHap.haplo object.
weights	A numeric vector providing HapAllele-specific weights.
batchsize	A numeric value controlling the number of haplotype alleles to be processed at a time (default = 500).
only.active.samples	A logical value specifying whether only active samples should be included in the output (default = TRUE).
only.active.alleles	A logical value specifying whether only active haplotype alleles should be included in the output (default = TRUE).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

Let $\mathbf{M}$ be the centered *N* x *H* matrix of HapGenotypes, where *N* is the number of individuals and *H* is the number of HapAlleles. The HapAllele covariance among individuals is computed as:

$$\mathbf{K} = q\mathbf{MDM}'$$

where $\mathbf{D} = diag(d_i)$, d_i is the weight of HapAllele *i* (default $d_i = 1$), and q is a scaling factor defined as $tr(\mathbf{MDM}')^{-1}M$. This is a generalization of the SNP-based genomic relationship matrix (VanRaden, 2008).

Value

The function returns a *n* x *n* matrix of HapAllele-based kinships, where *n* is the number of individuals.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Marco Milanesi <marco.milanesi.mm@gmail.com>

References

P. M. VanRaden. Efficient methods to compute genomic predictions. J. Dairy. Sci. 2008. 91:4414-4423.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
```

```
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# # Exclude minor alleles and singletons
# hapstats <- ghap.hapstats(haplo, ncores = 2)
# haplo <- ghap.subsethaplo(haplo,ids=haplo$id,alleles = hapstats$TYPE %in% c("REGULAR","MAJOR"))
#
# # Compute Kinship matrix
# K <- ghap.kinship(haplo, batchsize = 100)
```

`ghap.kinv` *Inverse of kinship matrix*

Description

Inversion of the haplotype covariance matrix

Usage

```
ghap.kinv(kinship,  method="nearPD", ncores=1, proven=NULL)
```

Arguments

`kinship`	A HapAllele-based kinship matrix, as supplied by `ghap.kinship`.
`method`	Inversion method: common inverse, inverse of the nearest positive definite matrix ("nearPD", default) and approximate inverse based on the Algorithm for Proven and Young animals ("APY").
`ncores`	Number of cores to be used for computations (default = 1). Only relevant for method "APY".
`proven`	Character vector with the names of proven subjects to be used as reference for method "APY".

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
# # Exclude minor alleles and singletons
# hapstats <- ghap.hapstats(haplo, ncores = 2)
# haplo <- ghap.subsethaplo(haplo,ids=haplo$id,alleles = hapstats$TYPE %in% c("REGULAR","MAJOR"))
#
# # Compute Kinship matrix
# K <- ghap.kinship(haplo, batchsize = 100)
#
#
# ### RUN ###
#
# # Common inverse
# Kinv <- ghap.kinv(K, method="common")
#
# # Inverse of nearest positive definite matrix
# Kinv <- ghap.kinv(K, method="nearPD")
#
# # APY inverse
# Kinv <- ghap.kinv(K, method="APY", proven=colnames(K)[1:500], ncores=2)
```

---

ghap.lmm *Linear mixed model*

Description

Linear mixed model fitting for fixed effects, random effects and variance components.

Usage

```
ghap.lmm(fixed, random, covmat = NULL, data, weights = NULL, family = "gaussian",
         REML = TRUE, verbose = TRUE)
```

Arguments

fixed	Formula describing the fixed effects part of the model, e.g. y ~ a + b + c ... If the model does not include any covariate simply state the response variable with an intercept, i.e. y ~ 1.
random	Formula describing the random effects part of the model, e.g., ~ x + w + z.
covmat	A list of covariance matrices for each group of random effects. If a matrix is not defined for a given group, an identity matrix will be used.
data	A dataframe containing the data.
weights	A numeric vector with weights for observations. These weights are treated as diagonal elements of the inverse weight matrix. If not supplied, the analysis is carried out assuming all observations are equally important.
family	A GLM family, see glm and family. Default is "gaussian".
REML	A logical value specifying whether the likelihood should be restricted regarding fixed effects (default = TRUE). Only relevant for the gaussian family with identity link function.
verbose	A logical value specifying whether log messages should be printed (default = TRUE).

Details

The function fits mixed models with correlated random effects using Cholesky factorization of covariance matrices. The default behaviour is to use the REstricted Maximum Likelihood (REML) algorithm implemented in lmer assuming a Gaussian family and an identity link function. However, regular Maximum Likelihood (ML) fit can be specified by setting the REML argument to FALSE. Additionally, generalized linear mixed models (GLMM) can be fit by specifying a different family and link function.

Value

The returned GHap.lmm object is a list with the following items:

fixed	A numeric vector containing the fixed effects.
random	A numeric vector containing the random effects.
vcp	A numeric vector with variance components.
residuals	A numeric vector containing residuals.
lme4	An object of class merMod.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

D. Bates et al. Fitting Linear Mixed-Effects Models Using lme4. J. Stat. Soft., 67:1-48.

A. I. Vazquez. Technical note: An R package for fitting generalized linear mixed models in animal breeding. J. Anim. Sci. 2010. 88, 497-504.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[haplo$pop %in% c("TSI","CEU")]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,rep(TRUE,times=haplo$nalleles))
# hapstats <- ghap.hapstats(haplo, ncores = 2)
# common <- hapstats$TYPE %in% c("REGULAR","MAJOR") &
#  hapstats$FREQ > 0.05 &
#  hapstats$FREQ < 0.95
# haplo <- ghap.subsethaplo(haplo,EUR.ids,common)
#
# #Compute relationship matrix
# K <- ghap.kinship(haplo, batchsize = 100)
#
# # Quantitative trait with 50% heritability
# # Unbalanced repeated measurements (0 to 30)
```

```
# # Two major haplotypes accounting for 50% of the genetic variance
# myseed <- 123456789
# set.seed(myseed)
# major <- sample(which(haplo$allele.in == TRUE),size = 2)
# g2 <- runif(n = 2, min = 0, max = 1)
# g2 <- (g2/sum(g2))*0.5
# sim <- ghap.simpheno(haplo, kinship = K, h2 = 0.5, g2 = g2, nrep = 30,
#                      balanced = FALSE, major = major, seed = myseed)
#
# #Fit model using REML
# model <- ghap.lmm(fixed = phenotype ~ 1, random = ~ individual,
#                   covmat = list(individual = K), data = sim$data)
#
# #Estimated heritability and repeatability
# model$vcp/sum(model$vcp)
#
# #True versus estimated breeding values
# plot(model$random$individual,sim$u,xlab="Estimated BV",ylab="True BV"); abline(0,1)
# summary(lm(sim$u ~ as.numeric(model$random$individual)))
```

`ghap.loadhaplo` *Load haplotype genotype data*

Description

This function loads HapGenotypes generated by `ghap.haplotyping` and converts them to a native GHap.haplo object.

Usage

```
ghap.loadhaplo(hapsamples.file, hapalleles.file, hapgenotypes.file, verbose = TRUE)
```

Arguments

`hapsamples.file`	Individual information file.
`hapalleles.file`	Haplotype alleles information file.
`hapgenotypes.file`	Haplotype genotype matrix file.
`verbose`	A logical value specifying whether log messages should be printed (default = TRUE).

Value

The returned GHap.haplo object is a list with components:

`nsamples`	An integer value for the sample size.
`nalleles`	An integer value for the number of haplotype alleles.
`nsamples.in`	An integer value for the number of active samples.
`nalleles.in`	An integer value for the number of active haplotype alleles.
`pop`	A character vector relating samples to populations. This information is obtained from the first column of the hapsamples file.
`id`	A character vector mapping genotypes to samples. This information is obtained from the second column of the hapsamples file.
`id.in`	A logical vector indicating active samples. By default, all samples are set to TRUE.
`chr`	A character vector mapping haplotype alleles to chromosomes. This information is obtained from the second column of the hapalleles file.
`block`	A character vector containing block names. This information is obtained from the first column of the hapalleles file.
`bp1`	A numeric vector with haplotype allele start positions. This information is obtained from the third column of the hapalleles file.
`bp2`	A numeric vector with haplotype allele end positions. This information is obtained from the fourth column of the hapalleles file.
`allele`	A character vector with haplotype allele identity. This information is obtained from the fifth column of the hapalleles file.
`allele.in`	A logical vector indicating active haplotype alleles. By default, all alleles are set to TRUE.
`genotypes`	A `big.matrix` object containing the haplotype genotype matrix

The input format is described in `ghap.haplotyping`.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Marco Milanesi <marco.milanesi.mm@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
```

```
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
#
# ### RUN ###
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
```

`ghap.loadphase` *Load phased genotype data*

Description

This function loads phased genotype data and converts them into a native GHap.phase object.

Usage

```
ghap.loadphase(samples.file, markers.file, phase.file, verbose = TRUE)
```

Arguments

`samples.file`	Individual information.
`markers.file`	Variant map information.
`phase.file`	Phased genotype matrix.
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The returned GHap.phase object is a list with components:

`chr`	A character value indicating chromosome identity. The current version of the package can handle only one chromosome at time.
`nsamples`	An integer value for the sample size.
`nmarkers`	An integer value for the number of markers.
`nsamples.in`	An integer value for the number of active samples.
`nmarkers.in`	An integer value for the number of active markers.

pop	A character vector relating chromosome alleles to populations. This information is obtained from the first column of the sample file.
id	A character vector mapping chromosome alleles to samples. This information is obtained from the second column of the sample file.
id.in	A logical vector indicating active chromosome alleles. By default, all chromosomes are set to TRUE.
marker	A character vector containing marker names. This information is obtained from the second column of the marker map file.
marker.in	A logical vector indicating active markers. By default, all markers are set to TRUE.
bp	A numeric vector with marker positions. This information is obtained from the third column of the marker map file.
A0	A character vector with reference alleles. This information is obtained from the fourth column of the marker map file.
A1	A character vector with alternative alleles. This information is obtained from the fifth column of the marker map file.
phase	A big.matrix object containing the phased genotype matrix.

The supported format is composed of three files with suffix:

- **.samples**: space-delimited file without header containing two columns: Population and ID. Please notice that the Population column serves solely for the purpose of grouping samples, so the user can define any arbitrary family/cluster/subgroup and use as a "population" tag.
- **.markers**: space-delimited file without header containing five columns: Chromosome, Marker, Position (in bp), Reference Allele (A0) and Alternative Allele (A1). Markers should be on a single chromosome and sorted by position. Repeated positions are tolerated, but a warning message is given when the data is loaded.
- **.phase**: space-delimited file without header containing the phased genotype matrix. The dimension of the matrix is expected to be *m x 2n*, where *m* is the number of markers and *n* is the number of individuals (i.e., two columns per individual, representing the two phased chromosome alleles). Alleles must be coded as 0 and 1. No missing values are allowed, since imputation is assumed to be part of the phasing procedure.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Marco Milanesi <marco.milanesi.mm@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
#
```

```
# ### RUN ###
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
```

---

ghap.maf *Compute marker minor allele frequencies*

Description

This function takes a GHap.phase object and computes the minor allele frequency for each marker.

Usage

```
ghap.maf(phase, only.active.samples = TRUE, only.active.markers = TRUE,
         ncores = 1)
```

Arguments

phase	A GHap.phase object.
only.active.samples	A logical value specifying whether only active samples should be used for calculations (default = TRUE).
only.active.markers	A logical value specifying whether only active markers should be included in the output (default = TRUE).
ncores	A numeric value specifying the number of cores to be used in parallel computing (default = 1).

Value

The function outputs a numeric vector of the same length of active markers containing minor allele frequencies based on the active samples.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Marco Milanesi <marco.milanesi.mm@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
#
# ### RUN ###
#
# # Calculate minor allele frequency
# maf <- ghap.maf(phase, ncores = 2)
```

`ghap.makefile` *Create example input files*

Description

Create example files to test the package.

Usage

```
ghap.makefile()
```

Details

This function copies the following example files to the current working directory:

human.phase

human.markers

human.samples

For details about the input format, see `ghap.loadphase`. The dataset was extracted from the reference phased data available at the IMPUTE2 (Howie et al., 2009) software website (https://mathgen.stats.ox.ac.uk/impute/impute_v2.html#reference).

The genotypes derive from the International HapMap Project Phase 3 (The International HapMap 3 Consortium, 2010), and comprise 1,011 subjects (from 11 populations) and 20,000 SNPs (randomly sampled from chromosome 2) mapped to the NCBI build 36 (hg18) assembly.

Author(s)

Marco Milanesi <marco.milanesi.mm@gmail.com>

References

B. N. Howie, P. Donnelly, and J. Marchini. A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. PLOS Genet. 2009. 5, e1000529.

The International HapMap 3 Consortium. Integrating common and rare genetic variation in diverse human populations. Nature. 2010. 467, 52-58.

Examples

```
# Copy the example data in the current working directory
ghap.makefile()
```

`ghap.mergehaplo` *Merging GHap.haplo objects*

Description

This function merges two GHap.haplo objects.

Usage

```
ghap.mergehaplo(haplo.1, haplo.2, type, only.active.markers = TRUE,
                only.active.samples = TRUE, verbose = TRUE)
```

Arguments

`haplo.1`	First GHap.haplo object.
`haplo.2`	Second GHap.haplo object.
`type`	A character value specifying the merging task to be performed. If type="HapAlleles", the two objects are assumed to have the same individuals but different HapAlleles. If type="individual", the two objects are assumed to have different individuals scored for the same HapAlleles.
`only.active.markers`	A logical value specifying whether only active markers should be included in the output (default = TRUE).
`only.active.samples`	A logical value specifying whether only active samples should be included in the output (default = TRUE).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The returned GHap.haplo object is a list with components specified by `ghap.loadhaplo`. Please notice that no duplicated individuals or HapAlleles (concatenation of *haplo$block* and *haplo$allele*) are allowed.

Author(s)

Marco Milanesi <marco.milanesi.mm@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# # Select CEU and ASW individuals
# CEU.ids <- haplo$id[which(haplo$pop=="CEU")]
# ASW.ids <- haplo$id[which(haplo$pop=="ASW")]
#
# # Randomly select a set of HapAlleles
# set.seed(1988)
# random.allele <- sample(x=c(TRUE, FALSE), size=haplo$nalleles, replace = TRUE)
#
# # Subset data
# haplo.CEU <- ghap.subsethaplo(haplo,ASW.ids,random.allele)
# haplo.ASW <- ghap.subsethaplo(haplo,CEU.ids,random.allele)
#
# # Merge haplo.CEU and haplo.ASW
# haplo.merge <- ghap.mergehaplo(haplo.1 = haplo.CEU, haplo.2 = haplo.ASW, type = "individual")
```

---

ghap.mergephase *Merging GHap.phase objects*

Description

This function merges two GHap.phase objects, provided they have the same set of markers.

Usage

```
ghap.mergephase(phase.1, phase.2, only.active.markers = TRUE,
                only.active.samples = TRUE, verbose = TRUE)
```

Arguments

`phase.1`	First GHap.phase object.
`phase.2`	Second GHap.phase object.
`only.active.markers`	A logical value specifying whether only active markers should be included in the output (default = TRUE).
`only.active.samples`	A logical value specifying whether only active samples should be included in the output (default = TRUE).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The returned GHap.phase object is a list with components specified by `ghap.loadphase`. Merging is allowed only if the two original GHap.phase objects have the same markers in the same order. Duplicated individuals are not tolerated.

Author(s)

Marco Milanesi <marco.milanesi.mm@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
#
# ### RUN ###
#
# # Select markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
#
```

```
# # Select ASW and CEU individuals
# ASW.ids <- unique(phase$id[phase$pop=="ASW"])
# CEU.ids <- unique(phase$id[phase$pop=="CEU"])
#
# # Subset data
# phase.ASW <- ghap.subsetphase(phase, ASW.ids, markers)
# phase.CEU <- ghap.subsetphase(phase, CEU.ids, markers)
#
# # Merge phase.ASW and phase.CEU
# phase.merge <- ghap.mergephase(phase.ASW, phase.CEU)
```

`ghap.outhaplo` *Output GHap.haplo object to a file*

Description

This function saves a GHap.haplo object to a text file using the same format specified by `ghap.haplotyping`.

Usage

```
ghap.outhaplo(haplo, outfile, only.active.markers = TRUE,
              only.active.samples = TRUE, verbose = TRUE)
```

Arguments

`haplo`	A GHap.haplo object.
`outfile`	A character value specifying the name for the output file.
`only.active.markers`	A logical value specifying whether only active markers should be included in the output (default = TRUE).
`only.active.samples`	A logical value specifying whether only active samples should be included in the output (default = TRUE).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The function outputs files with suffix *.hapalleles*, *.hapsamples* and *.hapgenotypes*, as specified by `ghap.haplotyping`.

Author(s)

Marco Milanesi <marco.milanesi.mm@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
# # Randomly select 500 individuals
# ids <- sample(x = haplo$id, size = 500, replace = FALSE)
#
# #Subset data
# haplo.sub <- ghap.subsethaplo(haplo,ids,haplo$allele.in)
#
#
# ### RUN ###
#
# # Output new GHap.haplo object
# ghap.outhaplo(haplo = haplo.sub, outfile = "humansub")
```

---

ghap.outphase *Output GHap.phase object to a file*

Description

This function saves a GHap.phase object to a text file using the same format specified by ghap.loadphase.

Usage

```
ghap.outphase(phase, outfile, only.active.markers = TRUE,
 only.active.samples = TRUE, verbose = TRUE)
```

Arguments

phase	A GHap.phase object.
outfile	A character value specifying the name for the output file.
only.active.markers	A logical value specifying whether only active markers should be included in the output (default = TRUE).
only.active.samples	A logical value specifying whether only active samples should be included in the output (default = TRUE).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The function outputs files with suffix *.markers*, *.samples* and *.phase*, as specified by ghap.loadphase.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
#
# ### RUN ###
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Output data
# ghap.outphase(phase, "example")
#
# # Reload
# phasesub <- ghap.loadphase("example.samples", "example.markers", "example.phase")
```

ghap.pca *Principal Components Analysis*

Description

PCA from a HapAllele-based kinship matrix.

Usage

```
ghap.pca(haplo, kinship, npc = 2)
```

Arguments

haplo	A GHap.haplo object.
kinship	A HapAllele-based kinship matrix, as supplied by ghap.kinship.
npc	Number of principal components to be retrieved (default = 2).

Value

The returned object is a list with items:

eigenvec	A data.frame containing the principal components of the kinship matrix.
eigenval	Vector with eigenvalues of the kinship matrix.
propvar	Vector with the proportion of variance explained by each principal component.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
```

```
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
# # Exclude minor alleles and singletons
# hapstats <- ghap.hapstats(haplo, ncores = 2)
# haplo <- ghap.subsethaplo(haplo,ids=haplo$id,alleles = hapstats$TYPE %in% c("REGULAR","MAJOR"))
#
# # Compute Kinship matrix
# K <- ghap.kinship(haplo, batchsize = 100)
#
#
# ### RUN ###
#
# # PCA analysis
# pca <- ghap.pca(haplo,K)
#
# # Plot
# plot(x=pca$eigenvec$PC1, y=pca$eigenvec$PC2, xlab="PC1", ylab="PC2", pch="")
# pop <- pca$eigenvec$POP
# pop.col <- as.numeric(as.factor(pop))
# pop <- sort(unique(pop))
# legend("bottomleft", legend = pop, col = 1:length(pop), pch = 1:length(pop), ncol = 3)
# points(x=pca$eigenvec$PC1, y=pca$eigenvec$PC2, pch = pop.col, col = pop.col, cex = 1.2)
```

`ghap.profile` *Haplotype allele profile*

Description

Given a data.frame of user-defined haplotype allele scores, compute individual profiles.

Usage

```
ghap.profile(score, haplo, only.active.samples = TRUE, ncores = 1)
```

Arguments

`score`	A data.frame containing columns: BLOCK, CHR, BP1, BP2, ALLELE, SCORE, CENTER and SCALE.
`haplo`	A GHap.haplo object.
`only.active.samples`	A logical value specifying whether calculations should be reported only for active samples (default = TRUE).
`ncores`	A numeric value specifying the number of cores to be used in parallel computing (default = 1).

Details

The profile for each individual is calculated as sum(b*(x-c)/s), where x is a vector of number of copies of each haplotype allele, c is a constant to center the genotypes (taken from the CENTER column of the score dataframe), s is a constant to scale the genotypes (taken from the SCALE column of the score dataframe), and b is a vector of user-defined scores for each haplotype allele (taken from the SCORE column of the score dataframe). If no centering or scaling is required, the user can set the CENTER and SCALE columns to 0 and 1, respectively. By default, if scores are provided for only a subset of the haplotype alleles, the missing alleles scores will be set to zero. This function has the same spirit as the profiling routine implemented in the *score* option in PLINK (Purcell et al., 2007; Chang et al., 2015).

Value

The function returns a data.frame with the following columns:

`POP`	Population ID.
`ID`	Individual name.
`PROFILE`	Individual profile.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com> Marco Milanesi <marco.milanesi.mm@gmail.com>

References

C. C. Chang et al. Second-generation PLINK: rising to the challenge of larger and richer datasets. Gigascience. 2015. 4, 7.

S. Purcell et al. PLINK: a tool set for whole-genome association and population-based linkage analyses. Am. J. Hum. Genet. 2007. 81, 559-575.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - randomly select 3000 markers with maf > 0.02
# maf <- ghap.maf(phase, ncores = 2)
# set.seed(1988)
# markers <- sample(phase$marker[maf > 0.02], 3000, replace = FALSE)
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
# rm(maf,markers)
#
# # Generate block coordinates based on windows of 10 markers, sliding 5 marker at a time
# blocks <- ghap.blockgen(phase, 10, 5, "marker")
```

```
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks, batchsize = 100, ncores = 2, freq = 0.05, outfile = "example")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("example.hapsamples", "example.hapalleles", "example.hapgenotypes")
#
#
# ### RUN ###
#
# # Create a score data.frame
# score <- NULL
# score$BLOCK <- haplo$block
# score$CHR <- haplo$chr
# score$BP1 <- haplo$bp1
# score$BP2 <- haplo$bp2
# score$ALLELE <- haplo$allele
# set.seed(1988)
# score$SCORE <- rnorm(length(score$ALLELE))
# score <- data.frame(score,stringsAsFactors = FALSE)
#
# # Compute profiles
# profile <- ghap.profile(score, haplo, ncores = 2)
```

`ghap.simpheno` *Quantitative trait simulation using real HapGenotype data*

Description

Simulates phenotypes from a quantitative trait with arbitrary major alleles.

Usage

```
ghap.simpheno(haplo, kinship, h2, g2, r2=0, nrep=1,
              balanced=TRUE, major=NULL, seed=NULL)
```

Arguments

`haplo`	A GHap.haplo object.
`kinship`	Covariance matrix to be used in polygenic effects simulation.
`h2`	A numeric value specifying the heritability.
`g2`	A numeric vector specifying the proportion of genetic variance explained by each major allele. The sum of the proportions must not exceed 1. In cases where the sum is less than 1, polygenic effects are simulated such that the remaining variance is uniformly distributed throughout the genome.
`r2`	A numeric value specifying the repeatability (default = 0). Only relevant if nrep > 1.

nrep	A numeric value specifying the number of repeated measures per subject.
balanced	A logical value specifying whether the output data should be balanced (default = TRUE). If balanced = FALSE, the number of repeated measures per subject will be heterogeneous, following a uniform distribution with minimum zero and maximum nrep. Only relevant if nrep > 1.
major	A numeric vector specifying the indices of the alleles to be considered as major.
seed	A numeric value used to set the random number generation state (default = NULL). This is useful for reproducibility of the results.

Details

The simulation considers the model:

$$\mathbf{y} = \mathbf{Zu} + \mathbf{Zp} + \mathbf{e}$$

where $\mathbf{u}$ is a vector of breeding values, $\mathbf{p}$ is a vector of permanent environmental effects, $\mathbf{Z}$ is an incidence matrix mapping $\mathbf{y}$ to $\mathbf{u}$ and $\mathbf{p}$, and $\mathbf{e}$ is the vector of residuals. Breeding values are assumed:

$$\mathbf{u} = \mathbf{Ha} + \mathbf{g}$$

where $\mathbf{a}$ is the vector of major allele effects, $\mathbf{H}$ is the centered matrix of major allele counts, and $\mathbf{g}$ is a vector of polygenic effects.

Value

The function returns a list with items:

h2	A numeric value specifying the heritability.
g2	A numeric vector specifying the proportion of genetic variance explained by each major HapAllele.
major	A numeric vector specifying the indices of the HapAlleles to be considered as major.
major.effect	A numeric vector containing the simulated major HapAllele effects.
u	A numeric vector containing breeding values.
p	A numeric vector containing permanent environmental effects. Suppressed if r2 = NULL and nrep = 1.
varu	A numeric value corresponding to the genetic variance.
varp	A numeric vector corresponding to the variance in permanent environmental effects. Suppressed if nrep = 1.
vare	A numeric value corresponding to the residual variance.
data	A data.frame containing columns: phenotype = a numeric vector containing the simulated phenotypes; individual = a character vector containing the IDs of the corresponding phenotypes.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[haplo$pop %in% c("TSI","CEU")]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,rep(TRUE,times=haplo$nalleles))
# hapstats <- ghap.hapstats(haplo, ncores = 2)
# common <- hapstats$TYPE %in% c("REGULAR","MAJOR") &
#  hapstats$FREQ > 0.05 &
#  hapstats$FREQ < 0.95
# haplo <- ghap.subsethaplo(haplo,EUR.ids,common)
#
# #Compute relationship matrix
# K <- ghap.kinship(haplo, batchsize = 100)
#
# # Quantitative trait with 50% heritability
# # Unbalanced repeated measurements (0 to 30)
# # Two major haplotypes accounting for 50% of the genetic variance
# myseed <- 123456789
# set.seed(myseed)
# major <- sample(which(haplo$allele.in == TRUE),size = 2)
# g2 <- runif(n = 2, min = 0, max = 1)
# g2 <- (g2/sum(g2))*0.5
```

```
# sim <- ghap.simpheno(haplo, kinship = K, h2 = 0.5, g2 = g2, nrep = 30,
#                      balanced = FALSE, major = major, seed = myseed)
```

---

ghap.subsethaplo *Subset GHap.hapo object*

Description

This function takes a list of alleles and individuals and subsets a GHap.haplo object.

Usage

```
ghap.subsethaplo(haplo, ids, alleles, verbose = TRUE)
```

Arguments

haplo	A GHap.haplo object.
ids	Character vector of individual names to keep.
alleles	Logical vector indicating alleles to be set to active (TRUE) or inactive (FALSE).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The returned GHap.haplo object (as described in the documentation for the ghap.loadhaplo function) is the same as the one used in the haplo argument. However, individuals not included in the ids vector are set to FALSE (i.e., inactivated) in the haplo$samples.in vector. The vector provided in the alleles argument replaces the haplo$allele.in vector in the new GHap.haplo object. This procedure avoids expensive subsetting operations by simply flagging which haplotype alleles and individuals should be used in downstream analyses.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
```

```
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks.mkr <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Generate matrix of haplotype genotypes
# ghap.haplotyping(phase, blocks.mkr, batchsize = 100, ncores = 2, outfile = "human")
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypes")
#
#
# ### RUN ###
#
# # Randomly select 500 individuals
# ids <- sample(x = haplo$id, size = 500, replace = FALSE)
#
# #Subset data
# haplo.sub <- ghap.subsethaplo(haplo,ids,haplo$allele.in)
```

`ghap.subsetphase` *Subset GHap.phase object*

Description

This function takes a list of markers and individuals and subsets a GHap.phase object.

Usage

```
ghap.subsetphase(phase, ids, markers, verbose = TRUE)
```

Arguments

`phase`	A GHap.phase object.
`ids`	Character vector of individual names to keep.
`markers`	Character vector of marker names to keep.
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The returned GHap.phase object (as described in the documentation for the `ghap.loadphase` function) is the same as the one used in the phase argument. However, individuals and markers not included in the provided vectors are set to FALSE (i.e., inactivated) in the phase$samples.in and phase$marker.in vectors, respectively. This procedure avoids expensive subsetting operations by simply flagging which markers and individuals should be used in downstream analyses.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# ghap.makefile()
#
# # Load data
# phase <- ghap.loadphase("human.samples", "human.markers", "human.phase")
#
#
# ### RUN ###
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.maf(phase, ncores = 2)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
```

Index