Package 'GHap'

September 21, 2020

Type Package

Title Genome-Wide Haplotyping

Version 2.0.0

Date 2020-08-12

Author Yuri Tani Utsunomiya, Marco Milanesi, Mario Barbato

Maintainer Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Description Haplotype calling from phased marker data. Given user-defined haplotype blocks (HapBlock), the package identifies the different haplotype alleles (HapAllele) present in the data and scores sample haplotype allele genotypes (HapGenotype) based on HapAllele dose (i.e. 0, 1 or 2 copies). The output is not only useful for analyses that can handle multi-allelic markers, but is also conveniently formatted for existing pipelines intended for bi-allelic markers. The package was first described in Bioinformatics by Utsunomiya et al. (2016, <doi:10.1093/bioinformatics/btw356>). Since the v2 release, the package provides functions for unsupervised and supervised detection of ancestry tracks. The methods implemented in these functions were described in an article published in Methods in Ecology and Evolution by Utsunomiya et al. (2020, <doi:10.1111/2041-210X.13467>).

License GPL (>= 2)

Imports parallel (>= 3.4.4), Matrix (>= 1.2-16), methods (>= 3.4.4), lme4 (>= 1.1-21), e1071 (>= 1.7-0.1), class (>= 7.3-15), data.table (>= 1.12.6)

VignetteBuilder R.rsp

Suggests R.rsp

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2020-09-21 08:50:08 UTC

R topics documented:

`ghap.anc2plink` *Convert ancestry tracks to PLINK binary*

Description

This function takes smoothed ancestry predictions obtained with the `ghap.ancsmooth` function and converts them to PLINK binary (bed/bim/fam) format.

Usage

```
ghap.anc2plink(phase, ancsmooth, ancestry, outfile, freq=c(0,1),
               missingness=1, only.active.samples=TRUE,
               only.active.markers=TRUE, batchsize=NULL,
               binary=TRUE, ncores=1, verbose=TRUE)
```

Arguments

phase	A GHap.phase object.
ancsmooth	A list containing smoothed ancestry classifications, such as supplied by the ghap.ancsmooth function.
ancestry	Character value indicating which ancestry to count at each observed marked site.
outfile	Character value for the output file name.
freq	A numeric vector of length 2 specifying the range of ancestry frequency to be included in the output. Default is c(0,1), which includes all marked sites.
missingness	A numeric value providing the missingness threshold to exclude marked sites with poor ancestry assignments (default = 1, with all sites retained).
only.active.samples	A logical value specifying whether only active samples should be included in the output (default = TRUE).
only.active.markers	A logical value specifying whether only active markers should be used for haplotyping (default = TRUE).
batchsize	A numeric value controlling the number of haplotype blocks to be processed and written to output at a time (default = 500).
binary	A logical value specfying whether the output file should be binary (default = TRUE).
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).

Details

The returned file mimics a standard PLINK (Purcell et al., 2007; Chang et al., 2015) binary file (bed/bim/fam), where counts 0, 1 and 2 represent the number of alleles assigned to the selected ancestry. For compatibility with PLINK, counts are coded as NN, NH and HH genotypes (N = NULL and H = haplotype allele), as if ancestry counts 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 character 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. Optionally, the user can use binary = FALSE to replace the bed file with a plain txt with ancestry counts.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window, slide = step, unit = "marker")
#
# # Supervised analysis with user-defined labels
# pops <- phase$pop
# labels <- pops
# labels[which(labels %in% c("CEU","TSI"))] <- "Europe"
# labels[which(labels %in% c("CHB","JPT","CHD"))] <- "Asia"
# labels[which(labels %in% c("YRI","LWK"))] <- "Africa"
# labels[which(labels %in% c("MKK","GIH","ASW","MEX"))] <- "Test"
# phase$pop <- labels
# train <- unique(phase$id[which(phase$pop != "Test")])
# prototypes <- ghap.anctrain(phase = phase, train = train,
```

```
#                              method = "supervised", ncores = 1)
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
# phase$pop <- pops
# anctracks <- ghap.ancsmooth(phase = phase, admix = hapadmix, ncores = 1)
#
# ### RUN ###
#
# # Export African ancestry in MEX to PLINK binary
# phase <- ghap.subsetphase(phase = phase,
#                           ids = unique(phase$id[which(phase$pop == "MEX")]),
#                           markers = phase$marker)
# ghap.anc2plink(phase = phase, ancsmooth = anctracks, ancestry = "Africa",
#                outfile = "mex_africa", ncores = 1, batchsize = 1000)
```

`ghap.ancmark` *Per marker ancestry proportions*

Description

Given smoothed ancestry predictions obtained with the `ghap.ancsmooth` function, per marker ancestry proportions are calculated across selected individuals.

Usage

```
ghap.ancmark(phase, ancsmooth, ids)
```

Arguments

`phase`	A GHap.phase object.
`ancsmooth`	A list containing smoothed ancestry classifications, such as supplied by the `ghap.ancsmooth` function.
`ids`	A character vector specifying which individuals to use for the calculations.

Details

This function takes smoothed ancestry classifications provided by the `ghap.ancsmooth` function and calculates, for each marker, the proportion of haplotypes carrying each ancestry label. The resulting output serve as a proxy for locus-specific ancestry proportions.

Value

The function returns a dataframes containing the following columns:

`CHR`	Chromosome name.
`MARKER`	Marker name.

BP	Marker position.
...	A number of columns (one for each ancestry label) giving the proportion of haplotypes carrying the respective ancestry label.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

See Also

ghap.ancsmooth

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window, slide = step, unit = "marker")
#
# # Supervised analysis with user-defined labels
# pops <- phase$pop
# labels <- pops
# labels[which(labels %in% c("CEU","TSI"))] <- "Europe"
# labels[which(labels %in% c("CHB","JPT","CHD"))] <- "Asia"
# labels[which(labels %in% c("YRI","LWK"))] <- "Africa"
# labels[which(labels %in% c("MKK","GIH","ASW","MEX"))] <- "Test"
# phase$pop <- labels
# train <- unique(phase$id[which(phase$pop != "Test")])
# prototypes <- ghap.anctrain(phase = phase, train = train,
#                             method = "supervised", ncores = 1)
```

```
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
# phase$pop <- pops
# anctracks <- ghap.ancsmooth(phase = phase, admix = hapadmix, ncores = 1)
#
# ### RUN ###
#
# # Get per marker ancestry proportions for MKK
# ancmark <- ghap.ancmark(phase = phase, ancsmooth = anctracks,
#                         ids = unique(phase$id[which(phase$pop == "MKK")]))
# plot(ancmark$BP/1e+6, ancmark$Africa, ylim = c(0,100), las=1, type="l", col="blue",
#      xlab = "Chromosome 2 (Mbp)", ylab = "Ancestry (%)", main = "Unsupervised")
# points(ancmark$BP/1e+6, ancmark$Asia, type="l", col="red")
# points(ancmark$BP/1e+6, ancmark$Europe, type="l", col="green")
# points(ancmark$BP/1e+6, ancmark$UNK, type="l", col="grey")
```

`ghap.ancplot` *Barplot of predictions of ancestry proportions*

Description

Given smoothed ancestry predictions obtained with the `ghap.ancsmooth` function, an admixture barplot is generated.

Usage

```
ghap.ancplot(ancsmooth, labels=TRUE, pop.ang=45, group.ang=0,
             colors=NULL, pop.order=NULL, sortby=NULL,
             use.unk=TRUE, legend=TRUE)
```

Arguments

`ancsmooth`	A list containing smoothed ancestry classifications, such as supplied by the `ghap.ancsmooth` function.
`labels`	A logic value indicating if population labels should be plotted (default = TRUE).
`pop.ang`	A numeric value representing the rotation of population labels in degrees (default = 45).
`group.ang`	A numeric value representing the rotation of group labels in degrees (default = 0).
`colors`	A character vector of colors to use for each ancestry label.
`pop.order`	A single character vector or a list of character vectors specifying the order of populations to plot (see details).
`sortby`	A character value indicating the ancestry label to use for sorting individuals within populations.
`use.unk`	A logical value indicating if the plot should be generated using missing values (default = TRUE).

`legend` A logical value indicating if a legend should be included to the plot (default = TRUE).

Details

This function takes smoothed ancestry classifications provided by the `ghap.ancsmooth` function and generates a traditional admixture/structure barplot. The argument pop.order allows the user to organize the displaying order of populations through a vector or a list. If a vector is provided, the populations are plotted following the order of elements within the vector. Otherwise, if a named list of vectors is provided, populations are first grouped by list elements and then displayed in the order they appear within their respective group vector.

The same data could be used to generate a circular barplot, for example with the **BITE** package.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

Milanesi, M., Capomaccio, S., Vajana, E., Bomba, L., Garcia, J.F., Ajmone-Marsan, P., Colli, L., 2017. BITE: an R package for biodiversity analyses. bioRxiv 181610. https://doi.org/10.1101/181610

See Also

`ghap.ancsmooth`, `ghap.ancmark`

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
```

```
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window, slide = step, unit = "marker")
#
# # Unsupervised analysis with best K
# prototypes <- ghap.anctrain(phase = phase, K = 3, ncores = 1)
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
# anctracks <- ghap.ancsmooth(phase = phase, admix = hapadmix, ncores = 1)
#
# ### RUN ###
#
# # List of population orders to plot
# pop.order <- vector("list",5)
# names(pop.order) <- c("Africa","Europe","East Asia","S. Asia","America")
# pop.order[[1]] <- c("YRI","LWK","MKK")
# pop.order[[2]] <- c("CEU","TSI")
# pop.order[[3]] <- c("CHB","JPT","CHD")
# pop.order[[4]] <- "GIH"
# pop.order[[5]] <- c("ASW","MEX")
#
# # Plot results
# ghap.ancplot(ancsmooth = anctracks, pop.order = pop.order)
```

---

ghap.ancsmooth *Smoothing of haplotype ancestry predictions*

Description

Given ancestry predictions obtained with the ghap.anctest or ghap.ancsvm functions, overlapping classifications are smoothed to refine the boundaries of recombination breakpoints.

Usage

```
ghap.ancsmooth(phase, admix, ncores=1, verbose=TRUE)
```

Arguments

phase	A GHap.phase object.
admix	A data frame containing ancestry classifications, such as supplied by the ghap.anctest or ghap.ancsvm functions.
ncores	A numeric value specifying the number of cores to be used in parallel computing (default = 1).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

This function takes results from ancestry classifications provided by the ghap.anctest or ghap.ancsvm functions and converts them into runs of ancestry. Since the classifiers assume exactly one ancestry per HapBlock, segments encompassing breakpoints are miss-classified as pertaining to a single origin, as opposed to a recombinant mixture of hybrid ancestry. When ghap.anctest/ghap.ancsvm are ran with overlapping HapBlocks, the smoothing function interrogates the ancestry of each overlapped segment by majority voting of all blocks containing it. After the ancestry of all segments have been resolved, contiguous sites sharing the same classification are converted into runs or segments of ancestry (i.e., ancestry tracks), which comprise the final output ('haplotypes' dataframe). These segments are then used to predict ancestry contributions ('proportions1' and 'proportions2' dataframes).

Value

The function returns three dataframes: 'proportions1', 'proportions2' and 'haplotypes'. The 'proportions1' dataframe contains the following columns:

POP	Original population label.
ID	Individual name.
...	A number of columns giving the predicted ancestry proportions.
UNK	The proportion of the genome without ancestry assignment.

The 'proportions2' dataframe is similar to 'proportions1', expect that ancestry contributions are recalibrated using only genome segments with ancestry assignments (therefore does not include the 'UNK' column). The 'haplotypes' dataframe contains the following columns:

POP	Original population label.
ID	Individual name.
HAP	Haplotype number.
CHR	Chromosome name.
BP1	Segment start position.
BP2	Segment end position.
SIZE	Segment size.
ANCESTRY	Predicted ancestry of the segment.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

See Also

ghap.anctrain, ghap.ancsvm, ghap.ancplot, ghap.ancmark

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window, slide = step, unit = "marker")
#
# # Unsupervised analysis with best K
# prototypes <- ghap.anctrain(phase = phase, K = 3, ncores = 1)
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
# anctracks <- ghap.ancsmooth(phase = phase, admix = hapadmix, ncores = 1)
#
# # Supervised analysis with user-defined labels
# pops <- phase$pop
# labels <- pops
# labels[which(labels %in% c("CEU","TSI"))] <- "Europe"
# labels[which(labels %in% c("CHB","JPT","CHD"))] <- "Asia"
# labels[which(labels %in% c("YRI","LWK"))] <- "Africa"
# labels[which(labels %in% c("MKK","GIH","ASW","MEX"))] <- "Test"
# phase$pop <- labels
# train <- unique(phase$id[which(phase$pop != "Test")])
# prototypes <- ghap.anctrain(phase = phase, train = train,
#                             method = "supervised", ncores = 1)
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
# phase$pop <- pops
# anctracks <- ghap.ancsmooth(phase = phase, admix = hapadmix, ncores = 1)
```

`ghap.ancsvm` *SVM-based predictions of haplotype ancestry*

Description

This function uses Support Vector Machines (SVM) to predict ancestry of haplotype alleles in test samples.

Usage

```
ghap.ancsvm(phase, blocks, test = NULL, train = NULL,
            cost = 1, gamma = NULL, tune = FALSE,
            only.active.samples = TRUE, only.active.markers = TRUE,
            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.
`test`	Character vector of individuals to test.
`train`	Character vector of individuals to use as reference samples.
`cost`	A numeric value specifying the C constant of the regularization term in the Lagrange formulation.
`gamma`	A numeric value specifying the gamma parameter of the RBF kernel (default = 1/blocksize).
`tune`	A logical value specfying if a grid search is to be performed for parameters (default = FALSE).
`only.active.samples`	A logical value specifying whether only active samples should be included in predictions (default = TRUE).
`only.active.markers`	A logical value specifying whether only active markers should be used for predictions (default = TRUE).
`ncores`	A numeric value specifying the number of cores to be used in parallel computing (default = 1).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Details

This function predicts haplotype allele ancestry using Support Vector Machines (SVM) together with a Gaussian Radial Basis Function (RBF) kernel. The user is required to specify the C constant of the regularization term in the Lagrange formulation (default cost = 1) and the gamma parameter (default gamma = 1/blocksize) of the RBF kernel.

Value

If ran with tune = FALSE, the function returns a dataframe with the following columns:

BLOCK	Block alias.
CHR	Chromosome name.
BP1	Block start position.
BP2	Block end position.
POP	Original population label.
ID	Individual name.
HAP1	Predicted ancestry of haplotype 1.
HAP2	Predicted ancestry of haplotype 2.

If tune = TRUE, the function returns a dataframe with the following columns:

cost	The candidate value of the C constant.
gamma	The canidate value of the gamma parameter.
accuracy	The percentage of correctly assigned ancestries.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

R. J. Haasl et al. Genetic ancestry inference using support vector machines, and the active emergence of a unique American population. Eur J Hum Genet. 2013. 21(5):554-62.

D. Meyer et al. e1071: Misc Functions of the Department of Statistics, Probability Theory Group (e1071). TU Wien. 2019 R Package Version 1.7-0.1. http://cran.r-project.org/web/packages/e1071/index.html.

See Also

svm, ghap.ancsmooth, ghap.ancplot, ghap.ancmark

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
```

```
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window, slide = step, unit = "marker")
#
# # User-defined labels
# pops <- phase$pop
# labels <- pops
# labels[which(labels %in% c("CEU","TSI"))] <- "Europe"
# labels[which(labels %in% c("CHB","JPT","CHD"))] <- "Asia"
# labels[which(labels %in% c("YRI","LWK"))] <- "Africa"
# labels[which(labels %in% c("MKK","GIH","ASW","MEX"))] <- "Test"
# phase$pop <- labels
# train <- unique(phase$id[which(phase$pop != "Test")])
# test <- unique(phase$id)
#
# # Tune supervised analysis
# ranblocks <- sample(x = 1:nrow(blocks), size = 5, replace = FALSE)
# tunesvm <- ghap.ancsvm(phase = phase, blocks = blocks[ranblocks,], train = train,
#                        gamma = 1/window*c(0.1,1,10), tune = TRUE, ncores = 1)
#
# # Supervised analysis with default parameters
# hapadmix <- ghap.ancsvm(phase = phase, blocks = blocks, test = test,
#                         train = train, ncores = 1)
# phase$pop <- pops
```

---

ghap.anctest *Prediction of haplotype ancestry*

Description

This function uses prototype alleles to predict ancestry of haplotypes in test samples.

Usage

```
ghap.anctest(phase, blocks, prototypes, test = NULL,
             only.active.samples = TRUE, only.active.markers = TRUE,
             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.
prototypes	A data frame containing prototype alleles, such as supplied by the ghap.anctrain function.
test	Character vector of individuals to test. All active individuals are used if this vector is not provided.
only.active.samples	A logical value specifying whether only active samples should be included in predictions (default = TRUE).
only.active.markers	A logical value specifying whether only active markers should be used for predictions (default = TRUE).
ncores	A numeric value specifying the number of cores to be used in parallel computing (default = 1).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

For each interrogated block, tested haplotypes are assigned to their nearest centroids (i.e., the pseudo-lineages with the smallest Euclidean distances).

Value

The function returns a dataframe with the following columns:

BLOCK	Block alias.
CHR	Chromosome name.
BP1	Block start position.
BP2	Block end position.
POP	Original population label.
ID	Individual name.
HAP1	Predicted ancestry of haplotype 1.
HAP2	Predicted ancestry of haplotype 2.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

See Also

ghap.anctrain, ghap.ancsmooth, ghap.ancplot, ghap.ancmark

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window, slide = step, unit = "marker")
#
# # Unsupervised analysis
# prototypes <- ghap.anctrain(phase = phase, K = 3, ncores = 1)
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
#
# # Supervised analysis with user-defined labels
# pops <- phase$pop
# labels <- pops
# labels[which(labels %in% c("CEU","TSI"))] <- "Europe"
# labels[which(labels %in% c("CHB","JPT","CHD"))] <- "Asia"
# labels[which(labels %in% c("YRI","LWK"))] <- "Africa"
# labels[which(labels %in% c("MKK","GIH","ASW","MEX"))] <- "Test"
# phase$pop <- labels
# train <- unique(phase$id[which(phase$pop != "Test")])
# prototypes <- ghap.anctrain(phase = phase, train = train,
#                             method = "supervised", ncores = 1)
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
# phase$pop <- pops
```

ghap.anctrain *Construction of prototype alleles*

Description

This function builds prototype alleles to be used in ancestry predictions.

Usage

```
ghap.anctrain(phase, train = NULL, method = "unsupervised",
              K = 2, iter.max = 10, nstart = 10, nmarkers = 5000,
              tune = FALSE, only.active.samples = TRUE,
              only.active.markers = TRUE, batchsize=NULL,
              ncores = 1, verbose = TRUE)
```

Arguments

	The following arguments are used by both the 'supervised' and 'unsupervised' methods:
	A GHap.phase object.
phase	Character vector of individuals to use as reference samples. All active individuals are used if this vector is not provided.
method	Character value indicating which method to use: 'supervised' or 'unsupervised' (default).
only.active.samples	A logical value specifying whether only active samples should be included in predictions (default = TRUE).
only.active.markers	A logical value specifying whether only active markers should be used for predictions (default = TRUE).
batchsize	A numeric value controlling the number of markers to be processed at a time (default = nmarkers/10).
ncores	A numeric value specifying the number of cores to be used in parallel computing (default = 1).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).
	The following arguments are only used by the 'unsupervised' method:
K	A numeric value specifying the number of clusters in K-means (default = 2). Proxy for the number of ancestral populations.
iter.max	A numeric value specifying the maximum number of iterations of the K-means clustering (default = 10).
nstart	A numeric value specifying the number of independent runs of the K-means clustering (default = 10).

nmarkers	A numeric value specifying the number of seeding markers to be used by the K-means clustering (default = 10).
tune	A logical value specfying if a Best K analysis should be performed (default = FALSE).

Details

This function builds prototype alleles (i.e., cluster centroids, representing lineage-specific allele frequencies) through two methods:

The 'unsupervised' method uses the K-means clustering algorithm to group haplotypes into K pseudo-lineages. A random sample of seeding markers (default value of nmarkers = 5000) is used to group all 2*nsamples haplotypes in a user-specified number of clusters (default value of K = 2). Then, for each interrogated block, prototype alleles are built for every cluster using the arithmetic mean of observed haplotypes initially assigned to that cluster. If train = NULL, the function uses all active haplotypes to build prototype alleles. If the user is working with a severely unbalanced data set (ex. one population with a large number of individuals and others with few individuals), it is recommended that a vector of individual names is provided via the train argument such that prototype alleles are built using a more balanced subset of the data.

The 'supervised' method works in a similar way, but skips the K-means algorithm and uses population labels present in the GHap.phase object as clusters.

Value

The function returns a dataframe with the first column giving marker names and remaining columns containing prototype alleles for each pseudo-lineage. If method 'unsupervised' is ran with tune = TRUE, the function returns the following list:

ssq	Within-cluster sum of squares for each value of K.
chindex	Calinski Harabasz Index for consecutive values of K.
pchange	Percent change in ssq for consecutive values of K.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

See Also

ghap.anctest, ghap.ancsmooth, ghap.ancplot, ghap.ancmark

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
```

```
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window, slide = step, unit = "marker")
#
# # BestK analysis
# bestK <- ghap.anctrain(phase = phase, K = 5, tune = TRUE, ncores = 1)
# plot(bestK$ssq, type = "b")
#
# # Unsupervised analysis with best K
# prototypes <- ghap.anctrain(phase = phase, K = 3, ncores = 1)
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
# anctracks <- ghap.ancsmooth(phase = phase, admix = hapadmix, ncores = 1)
#
# # Supervised analysis with user-defined labels
# pops <- phase$pop
# labels <- pops
# labels[which(labels %in% c("CEU","TSI"))] <- "Europe"
# labels[which(labels %in% c("CHB","JPT","CHD"))] <- "Asia"
# labels[which(labels %in% c("YRI","LWK"))] <- "Africa"
# labels[which(labels %in% c("MKK","GIH","ASW","MEX"))] <- "Test"
# phase$pop <- labels
# train <- unique(phase$id[which(phase$pop != "Test")])
# prototypes <- ghap.anctrain(phase = phase, train = train,
#                             method = "supervised", ncores = 1)
# hapadmix <- ghap.anctest(phase = phase, blocks = blocks, prototypes = prototypes,
#                          test = unique(phase$id), ncores = 1)
# phase$pop <- pops
# anctracks <- ghap.ancsmooth(phase = phase, admix = hapadmix, ncores = 1)
```

---

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,
           batchsize = NULL, ncores = 1, verbose = TRUE)
```

Arguments

response	A vector of phenotypes. The vector must be named and all names must be present in the GHap.haplo object.
haplo	A 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.
gc	A logical value specifying whether genomic control should be performed (default = TRUE). Currently, this option does not take effect on HapBlocks.
only.active.alleles	A logical value specifying whether calculations should be reported only for active haplotype alleles (default = TRUE).
batchsize	A numeric value controlling the number of HapAlleles to be processed at a time (default = nalleles/10).
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).

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:

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

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

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

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). An alternative to these methods is to use polygenic effects as response instead of residuals. However, this can lead to a higher false-positive rate (Ekine et al., 2014). Finally, if the analysis is intended to screen the genome for recovering effects that are not captured by single markers, a more traditional kinship based on single markers instead of HapAlleles could be used to fit the linear mixed model.

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.

C. C. Ekine et al. Why breeding values estimated using familial data should not be used for genome-wide association studies. G3. 2014. 4:341-347.

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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[which(haplo$pop %in% c("TSI","CEU"))]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,haplo$allele.in)
# hapstats <- ghap.hapstats(haplo, ncores = 1)
# common <- hapstats$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, ncores = 1)
#
# # 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 = 1)
#
# #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 = 1, weights = w)
#
# #HapAllele GWAS using residuals as response
# pheno <- model$residuals
# names(pheno) <- sim$data$individual
# gwas3 <- ghap.assoc(response = pheno, haplo = haplo, ncores = 1)
#
# #Plot results
# plot(gwas1$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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
#
# ### 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 kbp sliding 100 kbp at a time
# blocks.kb <- ghap.blockgen(phase, windowsize = 100, slide = 100, unit = "kbp")
```

`ghap.blockstats` *HapBlock statistics*

Description

Generate HapBlock summary statistics from pre-computed HapAlleles statistics.

Usage

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

Arguments

`hapstats`	A data.frame containing HapAllele statistics, as generated by the `ghap.hapstats` function.
`ncores`	A numeric value specifying the number of cores to be used in parallel computing (default = 1).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Details

For each HapBlock, the function counts the number of unique HapAlleles and computes the expected heterozygosity $1 - \sum {p_i}^2$, where p_i is the frequency of HapAllele *i*. Please notice that when HapAlleles 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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
# Subset
# ids <- haplo$id[which(haplo$pop == "CEU")]
# haplo <- ghap.subsethaplo(haplo = haplo, ids = ids, alleles = haplo$allele.in)
#
# #Compute haplotype statistics
# hapstats <- ghap.hapstats(haplo, ncores = 1)
#
#
# ### RUN ###
#
# #Compute block statistics
# blockstats <- ghap.blockstats(hapstats, ncores = 1)
```

`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,
        only.active.alleles = TRUE, batchsize = NULL, ncores = 1, verbose = TRUE)
```

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., inverse genomic kinship) matrix for GEBVs.
gebvsweights	A numeric vector providing individual-specific weights.
haploweights	A numeric vector providing HapAllele-specific weights.
only.active.alleles	A logical value specifying whether only active HapAlleles should be included in the calculations (default = TRUE).
batchsize	A numeric value controlling the number of HapAlleles to be processed at a time (default = nalleles/10).
ncores	A numeric value specifying the number of cores to be used in parallel computing (default = 1).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

The function uses the equation:

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

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).

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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[which(haplo$pop %in% c("TSI","CEU"))]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,haplo$allele.in)
# hapstats <- ghap.hapstats(haplo, ncores = 1)
# common <- hapstats$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, ncores = 1)
#
# # 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$individual)
# gebvsw <- gebvsw + mean(gebvsw)
# gebvsw <- gebvsw[names(gebvs)]
# Kinv <- ghap.kinv(K)
# gwas.blup <- ghap.blup(gebvs = gebvs, haplo = haplo, gebvsweights = gebvsw,
#                        ncores = 1, invcov = Kinv)
# plot(gwas.blup$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$VAR*nrow(gwas.blup)
# K2 <- ghap.kinship(haplo=haplo, weights = w, ncores=4)
# Kinv2 <- ghap.kinv(K2)
# gwas.blup2 <- ghap.blup(gebvs = gebvs, haplo = haplo, invcov = Kinv2, ncores = 1,
#                         gebvsweights = gebvsw, haploweights = w)
# plot(gwas.blup2$BP1/1e+6,gwas.blup2$pVAR*100,pch=20,
#      xlab="Position (in Mb)",ylab="Variance explained (%)")
# abline(v=haplo$bp1[major]/1e+6)
```

`ghap.compress` *Compress phased genotype data*

Description

This function takes phased genotype data and converts them into a compressed binary format.

Usage

```
ghap.compress(input.file = NULL, out.file, samples.file = NULL,
              markers.file = NULL, phase.file = NULL, verbose = TRUE)
```

Arguments

	If all input files share the same prefix, the user can use the following shortcut options:
	Prefix for input files.
output.file	Output file name.
	For backward compatibility, the user can still point to input files separately:
samples.file	Individual information.
markers.file	Variant map information.
phase.file	Phased genotype matrix.
	To turn compression progress-tracking on or off please use:
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

The supported input 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 not tolerated.
- **.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 or 1. No missing values are allowed, since imputation is assumed to be part of the phasing procedure.

The function outputs a binary file with suffix **.phaseb**. Each allele is stored as a bit in that file. Bits for any given marker are arranged in a sequence of bytes. Since each marker requires storage of 2*nsamples bits, the number of bytes consumed by a single marker in the output file is ceiling(2*nsamples). If the number of alleles is not a multiple of 8, bits in the remainder of the last byte are filled with 0. All functions in GHap were carefully designed to decode the bytes of a marker in such a way that trailing bits are ignored if present.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

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

```
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
#
# # Compress phase data using prefix
# ghap.compress(input.file = "human", out.file = "human")
#
# # Compress phase data using file names
# ghap.compress(samples.file = "human.samples",
#               markers.file = "human.markers",
#               phase.file = "human.phase",
#               out.file = "human")
```

---

ghap.fast2phase *Convert fastPHASE data into the GHap phase format*

Description

This function takes phased genotype data in fastPHASE format and converts them into a GHap plan and compressed binary format.

Usage

```
ghap.fast2phase(input.files = NULL, switchout.files = NULL,
                map.files = NULL, out.file, verbose = TRUE)
```

Arguments

	If all input files share the same prefix, the user can use the following shortcut options:
	Character vector with the list of prefixes for input files.
output.files	Character value for the output file name.
	The user can also opt to point to input files separately:
switchout.files	
	Character vector containing the list of fastPHASE files
map.files	Character vector containing the list of map files.
	To turn conversion progress-tracking on or off please use:
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

Currently this function handles only _switch.out files from fastPHASE v1.4.0. The map files should contain the following 5 space-delimited columns: chromosome, marker, position, allele 0 and allele 1.

Author(s)

Mario Barbato <mario.barbato@unicatt.it>, Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

Scheet, P., Stephens, M., 2006. A Fast and Flexible Statistical Model for Large-Scale Population Genotype Data: Applications to Inferring Missing Genotypes and Haplotypic Phase. Am. J. Hum. Genet. 78, 629–644. https://doi.org/10.1086/502802.

See Also

ghap.compress, ghap.loadphase, ghap.oxford2phase, ghap.vcf2phase

---

ghap.freq *Compute marker allele frequencies*

Description

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

Usage

```
ghap.freq(phase, type = "maf", only.active.samples = TRUE, only.active.markers = TRUE,
          batchsize = NULL, ncores = 1, verbose = TRUE)
```

Arguments

phase	A GHap.phase object.
type	A character value indicating which allele frequency to compute. Valid options are minor allele frequency ('maf', default), frequency of allele 0 ('A0') and frequency of allele 1 ('A1').
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).
batchsize	A numeric value controlling the number of markers to be processed at a time (default = nmarkers/10).
ncores	A numeric value specifying the number of cores to be used in parallel computing (default = 1).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The function outputs a numeric vector of the same length of active markers containing 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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
#
# ### RUN ###
#
# # Calculate minor allele frequency
# maf <- ghap.freq(phase, type = 'maf', ncores = 1)
```

---

ghap.fst *Haplotype-based 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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
```

```
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypesb")
#
#
# ### 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$allele.in)
# CHB.hapstats <- ghap.hapstats(haplo,ncores = 1)
# haplo <- ghap.subsethaplo(haplo,CEU.ids,haplo$allele.in)
# CEU.hapstats <- ghap.hapstats(haplo,ncores = 1)
# haplo <- ghap.subsethaplo(haplo,c(CHB.ids,CEU.ids),haplo$allele.in)
# TOT.hapstats <- ghap.hapstats(haplo,ncores = 1)
# haplo <- ghap.subsethaplo(haplo,haplo$id,haplo$allele.in)
#
# # Compute haplotype block statistics for each group
# CHB.blockstats <- ghap.blockstats(CHB.hapstats, ncores = 1)
# CEU.blockstats <- ghap.blockstats(CEU.hapstats, ncores = 1)
# TOT.blockstats <- ghap.blockstats(TOT.hapstats, ncores = 1)
#
# # Calculate Fst
# fst <- ghap.fst(CHB.blockstats, CEU.blockstats, TOT.blockstats)
#
# # Plot results
# top.fst <- fst[which(fst$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$FST, na.rm=TRUE), "EDAR", col="red")
# CEU.hapstats[which(CEU.hapstats$BLOCK == top.fst$BLOCK & CEU.hapstats$FREQ > 0),1:9]
# CHB.hapstats[which(CHB.hapstats$BLOCK == top.fst$BLOCK & CHB.hapstats$FREQ > 0),1:9]
```

---

ghap.hap2plink *Convert haplotype allele counts to PLINK binary*

Description

This function takes a HapGenotypes matrix (as generated with the ghap.haplotyping function) and converts it to PLINK binary (bed/bim/fam) format.

Usage

```
ghap.hap2plink(haplo, outfile)
```

Arguments

haplo	A GHap.haplo object.
outfile	A character value specifying the name used for the .bed, .bim and .fam output files.

Details

The returned file mimics a standard PLINK (Purcell et al., 2007; Chang et al., 2015) binary file (bed/bim/fam), where HapAllele counts 0, 1 and 2 are recoded as NN, NH and HH genotypes (N = NULL and H = haplotype allele), as if HapAlleles 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 character 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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
```

```
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo("human.hapsamples", "human.hapalleles", "human.hapgenotypesb")
#
#
# ### RUN ###
#
# # Convert to plink
# ghap.hap2plink(haplo, outfile = "human")
```

`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,
                 only.active.samples=TRUE, only.active.markers=TRUE,
                 batchsize=NULL, binary=TRUE, 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).

`only.active.samples`
: A logical value specifying whether only active samples should be included in the output (default = TRUE).

`only.active.markers`
: A logical value specifying whether only active markers should be used for haplotyping (default = TRUE).

`batchsize`
: A numeric value controlling the number of haplotype blocks to be processed and written to output at a time (default = nblocks/10).

`binary`
: A logical value specfying whether the output file should be binary (default = TRUE).

`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**: if binary = FALSE, a 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.
- **.hapgenotypesb**: if binary = TRUE (default), the same matrix as described above compressed into bits. For seamless compatibility with softwares that use PLINK binary files, the compression is performed using the SNP-major bed format.

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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
```

```
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
#
# ### RUN ###
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
```

`ghap.hapstats` *Haplotype allele statistics*

Description

Summary statistics for HapAlleles.

Usage

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

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).
`batchsize`	A numeric value controlling the number of HapAlleles to be processed at a time (default = nalleles/10).
`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).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

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 HapAllele: "SINGLETON" = single allele of its block; "ABSENT" = the frequency of the allele is 0; "MINOR" = the least frequent allele of its block (in the case of ties, only the first allele is marked); "MAJOR" = the most frequent allele of its block (ties are also resolved by marking the first allele); "REGULAR" = the allele 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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
```

```
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
#
# ### RUN ###
#
# Subset
# ids <- haplo$id[which(haplo$pop == "CEU")]
# haplo <- ghap.subsethaplo(haplo = haplo, ids = ids, alleles = haplo$allele.in)
#
# #Compute haplotype statistics
# hapstats <- ghap.hapstats(haplo, ncores = 1)
```

`ghap.hslice` *Get a slice of the haplo object*

Description

This function parses the binary HapGenotypes matrix and returns the slice as an R matrix.

Usage

```
ghap.hslice(haplo, ids, alleles, index=FALSE, lookup=NULL,
            ncores=1, verbose=TRUE)
```

Arguments

`haplo`	A GHap.haplo object.
`ids`	A character or numeric vector indicating individuals to parse.
`alleles`	A numeric vector indicating alleles to parse.
`index`	A logical value specfying if values provided for ids are indices (see details).
`lookup`	A character vector containing the look up table to decode the binary file. Only meaningful for developers.
`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).

Details

This function parses the file with suffix *.hapgenotypesb* and returns an R matrix with the requested list of haplotype alleles and individuals. The argument index allows the user to specify individuals either by name (index = FALSE) or by indices as stored in the GHap.haplo object (index = TRUE). HapAlleles can only by parsed via indices. The lookup argument also allows the user to provide a vector containing the decoding scheme to translate bits into integers in R. However, this argument should only be used by developers.

Value

An R matrix with haplotype alleles in rows and individuals in columns.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
# ### RUN ###
#
# ids <- sample(x = haplo$id, size = 10, replace = F)
# alleles <- sample(x = 1:haplo$nalleles, size = 10, replace = F)
# X <- ghap.hslice(haplo = haplo, ids = ids, alleles = alleles, index = F, ncores = 1)
```

ghap.karyoplot *Individual chromosome painting*

Description

Given smoothed ancestry predictions obtained with the ghap.ancsmooth function and the name of the indivual, an individual karyotype plot is generated.

Usage

```
ghap.karyoplot(ancsmooth, ids=NULL, colors=NULL, chr=NULL,
               chr.line=10, plot.line=25, chr.ang=45, las=0)
```

Arguments

ancsmooth	A list containing smoothed ancestry classifications, such as supplied by the ghap.ancsmooth function.
ids	A character vector of individual(s) to plot. If NULL all the individuals will be plotted (default = NULL).
colors	A character vector of colors to use for each ancestry label (default = NULL).
chr	A vector with the chromosome(s) to plot. If NULL, all the chromosomes will be plotted (default = NULL).
chr.line	A numeric value representing the number of chromosomes per plot line (default = 10).
plot.line	A numeric value representing the distance of horizontal guide (default = 25).
chr.ang	A numeric value representing the rotation of chromosome labels in degrees (default = 45).
las	A numeric value representing the las of y-axes (default = 0).

Details

This function takes smoothed ancestry classifications provided by the ghap.ancsmooth function and "paint" the chomosomes of one individual using the ancestry proportions. One or more individuals could be plotted in separated graph.

Author(s)

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

See Also

ghap.anctrain, ghap.anctest, ghap.ancsvm, ghap.ancsmooth, ghap.ancmark, ghap.ancplot

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data to the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window, slide = step, unit = "marker")
#
# # Unsupervised analysis with best K
# admix.kmeans <- ghap.ancestry(phase = phase, blocks = blocks, test = unique(phase$id),
#                               method = "unsupervised", K = 3, ncores = 1)
# out.kmeans <- ghap.ancsmooth(phase = phase, admix = admix.kmeans, ncores = 1)
#
# # Training and test set for supervised analysis
# pops <- phase$pop
# names(pops) <- phase$id
# labels <- pops
# labels[which(labels %in% c("CEU","TSI"))] <- "Europe"
# labels[which(labels %in% c("CHB","JPT","CHD"))] <- "Asia"
# labels[which(labels %in% c("YRI","LWK"))] <- "Africa"
# labels[which(labels %in% c("MKK","GIH","ASW","MEX"))] <- "Test"
# phase$pop <- labels
# train <- unique(phase$id[which(phase$pop != "Test")])
# test <- unique(phase$id[which(phase$pop == "Test")])
#
# # Supervised analysis with best parameters
# admix.svm <- ghap.ancestry(phase = phase, blocks = blocks, train = train, test = test,
#                            method = "supervised", fit.train = TRUE, ncores = 1)
# phase$pop <- pops
# admix.svm$POP <- pops[admix.svm$ID]
# out.svm <- ghap.ancsmooth(phase = phase, admix = admix.svm, ncores = 1)
#
```

```
# ### RUN ###
#
# # Select some individuals
# ids <- c("NA19835", "NA12003", "NA18749", "NA17969", "NA21108",
#          "NA18998", "NA19394", "NA19678", "NA21453", "NA20529", "NA19223")
#
# # Plot karyoplot
# ghap.karyoplot(ancsmooth = out.kmeans, ids = ids[1], chr.line = 11,
#                plot.line = 50, las=1, chr=NULL)
# ghap.karyoplot(ancsmooth = out.svm, ids = ids[1], chr.line = 11,
#                plot.line = 50, las=1)
```

`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=NULL, batchsize=NULL, only.active.samples=TRUE,
             only.active.alleles=TRUE, ncores=1, verbose=TRUE)
```

Arguments

`haplo`	A GHap.haplo object.
`weights`	A numeric vector providing HapAllele-specific weights.
`batchsize`	A numeric value controlling the number of HapAlleles to be processed at a time (default = nalleles/10).
`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).
`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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
```

```
#
# ### RUN ###
#
# # Exclude minor alleles and singletons
# hapstats <- ghap.hapstats(haplo, ncores = 1)
# haplo <- ghap.subsethaplo(haplo,ids=haplo$id, alleles = hapstats$TYPE %in% c("REGULAR","MAJOR"))
#
# # Compute Kinship matrix
# K <- ghap.kinship(haplo, ncores = 1)
```

---

ghap.kinv *Inverse of kinship matrix*

Description

Inversion of the haplotype covariance matrix

Usage

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

Arguments

kinship	A HapAllele-based kinship matrix, as supplied by ghap.kinship.
method	Inversion method: common inverse ("common"), inverse of the nearest positive definite matrix ("nearPD", default) or 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".
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
```

```
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
# # Subset
# ids <- haplo$id[which(haplo$pop == "CEU")]
# haplo <- ghap.subsethaplo(haplo = haplo, ids = ids, alleles = haplo$allele.in)
#
# # Exclude minor alleles and singletons
# hapstats <- ghap.hapstats(haplo, ncores = 1)
# haplo <- ghap.subsethaplo(haplo,ids=ids, alleles = hapstats$TYPE %in% c("REGULAR","MAJOR"))
#
# # Compute Kinship matrix
# K <- ghap.kinship(haplo, ncores = 1)
#
#
# ### 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=1)
```

---

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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[which(haplo$pop %in% c("TSI","CEU"))]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,haplo$allele.in)
# hapstats <- ghap.hapstats(haplo, ncores = 1)
# common <- hapstats$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, ncores = 1)
#
# # 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)
#
#
# ### RUN ###
#
# #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(input.file = NULL, hapsamples.file = NULL,
               hapalleles.file = NULL, hapgenotypesb.file = NULL,
               verbose = TRUE)
```

Arguments

`input.file`	Prefix for input files.
`hapsamples.file`	Individual information file.
`hapalleles.file`	Haplotype alleles information file.
`hapgenotypesb.file`	Binary 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 character value giving the pathway to the binary 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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# ### RUN ###
#
```

```
# # Load haplotype genotypes using prefix
# haplo <- ghap.loadhaplo(input.file = "human")
#
# # Load haplotype genotypes using file names
# haplo <- ghap.loadhaplo(hapsamples.file = "human.hapsamples",
#                         hapalleles.file = "human.hapalleles",
#                         hapgenotypesb.file = "human.hapgenotypesb")
```

`ghap.loadphase` *Load binary phased genotype data*

Description

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

Usage

```
ghap.loadphase(input.file = NULL, samples.file = NULL, markers.file = NULL,
               phaseb.file = NULL, verbose = TRUE)
```

Arguments

	If all input files share the same prefix, the user can use the following shortcut option:
	Prefix for input files.
	For backward compatibility, the user can still point to input files separately:
`samples.file`	Individual information.
`markers.file`	Variant map information.
`phaseb.file`	Binary phased genotype matrix, such as supplied by the `ghap.compress` function.
	To turn loading progress-tracking on or off please use:
`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 vector indicating chromosome identity for each marker.
`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 character value giving the pathway to the binary phased genotype matrix.

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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# ### RUN ###
#
# # Load data using prefix
# phase <- ghap.loadphase(input.file = "human")
#
# # Load data using file names
# phase <- ghap.loadphase(samples.file = "human.samples",
#                         markers.file = "human.markers",
#                         phaseb.file = "human.phaseb")
```

`ghap.makefile` *Create example input files*

Description

Create example files to test the package.

Usage

```
ghap.makefile(verbose = TRUE)
```

Arguments

`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Details

This function downloads the following example files to the R temporary directory (requires internet connection):

human.phase
human.markers
human.samples

For details about the format of these files, see `ghap.compress`. 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. The source link is (https://mathgen.stats.ox.ac.uk/impute/impute_v2.html#reference)

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>
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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
```

`ghap.manhattan` *Manhattan plot*

Description

Generate a Manhattan plot from a dataframe.

Usage

```
ghap.manhattan(data, chr, bp, y, colors=NULL, type="p", pch=20,
               cex=1, lwd=1, ylim=NULL, ylab="", xlab="", main="",
               backcolor="#F5EFE780", chr.ang=0, hlines = NULL,
               hcolors = NULL, hlty = 1, hlwd = 1)
```

Arguments

`data`	A data.frame containing the data to be plotted.
`chr`	A character value with the name of the column containing chromosome labels.
`bp`	A character value with the name of the column containing base pair positions.
`y`	A character value with the name of the column containing the variable to be plotted in the y axis.
`colors`	A character value containing colors to be used for chromosomes.
`type`	What type of plot should be drawn (default = "p"). See `plot`.
`pch`	Either an integer specifying a symbol or a single character to be used as the default in plotting points (default = 20). See `points` for possible values and their interpretation.
`cex`	A numeric value for the relative point size (default = 1).
`lwd`	A numeric value for the line width (default = 1). Only meaningful for type="l".
`ylim`	A numeric vector of size 2 containing the lower and upper limits of the y-axis.
`ylab`	A chracter value for the y-axis label.
`xlab`	A chracter value for the x-axis label.
`main`	A chracter value for the plot title.
`backcolor`	The background color.
`chr.ang`	A numeric value representing the rotation of chromosome labels in degrees (default = 0).
`hlines`	A numeric vector containing y-axis positions for horizontal lines.
`hcolors`	A character vector containing colors for the horizontal lines.
`hlty`	A numeric vector containing types for horizontal lines.
`hlwd`	A numeric vector for the relative width of vertical lines.

Details

This function takes a dataframe of genomic positions and generates a Manhattan plot. The chromosome column must be a vector of factors (the order of the chromosomes will be displayed according to the order of the factor levels).

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# ### RUN ###
#
# # Generate some data
# set.seed(1988)
# genome <- c(1:22,"X")
# chr <- rep(genome, each = 1000)
# bp <- rep(1:1000, times = length(genome))
# y <- abs(rnorm(n = length(bp)))
# y <- 1 - (pnorm(y) - pnorm(-y))
# y <- -log10(y)
# y[10300:10350] <- sort(runif(n = 51, min = 0, max = 10))
# mydata <- data.frame(chr,bp,y)
# mydata$chr <- factor(x = mydata$chr, levels = genome, labels = genome)
#
# # Minimal plot
# ghap.manhattan(data = mydata, chr = "chr", bp = "bp", y = "y")
#
# # Customization example
# ghap.manhattan(data = mydata, chr = "chr", bp = "bp", y = "y",
#                main = "Genome-wide association analysis", ylab = "-log10(p)",
#                xlab = "Chromosome", chr.ang = 90, backcolor = "white", type = "l",
#                hlines = c(6,8), hlty = c(2,3), hcolors = c(1,2), hlwd = c(1,2))
```

---

ghap.oxford2phase *Convert Oxford data into GHap phase*

Description

This function takes phased genotype data in Oxford HAPS/SAMPLES format and converts them into the GHap phase format.

Usage

```
ghap.oxford2phase(input.files = NULL, haps.files = NULL, sample.files = NULL,
                  out.file, verbose = TRUE)
```

Arguments

	If all input files share the same prefix, the user can use the following shortcut options:
	Character vector with the list of prefixes for input files.
output.files	Character value for the output file name.
	The user can also opt to point to input files separately:
haps.files	Character vector containing the list of Oxford HAPS files.
sample.files	Character vector containing the list of Oxford SAMPLES files.
	To turn conversion progress-tracking on or off please use:
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

The Oxford HAPS/SAMPLE format output of widely used phasing software such as SHAPEIT2 (O'Connell et al., 2014) or Eagle (Loh et al., 2106) is here manipulated to obtain the GHap phase format.

Author(s)

Mario Barbato <mario.barbato@unicatt.it>, Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

R-R. Loh P-R et al. Reference-based phasing using the Haplotype Reference Consortium panel. Nat Genet. 2016. 48(11):1443-1448.

J. O'Connell et al. A general approach for haplotype phasing across the full spectrum of relatedness. PLOS Genet. 2014. 10:e1004234.

See Also

ghap.compress, ghap.loadphase, ghap.fast2phase, ghap.vcf2phase

Examples

```
# # Build toy example data (2 chromosomes, 5 samples and 5 markers per chromosome)
# samples <- data.frame(V1 = c("ID_1","0","POP1","POP1","POP1","POP2","POP2"),
#                       V2 = c("ID_2","0","ID1","ID2","ID3","ID4","ID5"),
#                       V3 = c("missing","0","0","0","0","0","0"),
#                       stringsAsFactors = FALSE)
# bases <- c("A","T","C","G")
# for(i in 1:2){
#   map <- data.frame(V1 = rep(i, times = 5),
#                     V2 = paste("C",i,"P",1:5,sep=""),
#                     V3 = 1:5,
#                     V4 = sample(x = bases, size = 5, replace = TRUE),
#                     V5 = sample(x = bases, size = 5, replace = TRUE))
```

```
#  phase <- matrix(data = sample(x = c(0,1), size = 2*5*5, replace = TRUE), nrow = 5, ncol = 2*5)
#   haps <- cbind(map,phase)
#   write.table(file = paste("example_chr",i,".sample",sep=""), x = samples,
#               quote = FALSE, sep = " ", row.names = FALSE, col.names = FALSE)
#   write.table(file = paste("example_chr",i,".haps",sep=""), x = haps,
#               quote = FALSE, sep = " ", row.names = FALSE, col.names = FALSE)
# }
#
# # Convert Oxford HAPS/SAMPLES to GHap phase using prefix
# ghap.oxford2phase(input.files = paste("example_chr",1:2,sep=""),
#                   out.file = "example")
#
# # Convert Oxford HAPS/SAMPLES to GHap phase using file names
# ghap.oxford2phase(haps.files = paste("example_chr",1:2,".haps",sep=""),
#                   sample.files = paste("example_chr",1:2,".sample",sep=""),
#                   out.file = "example")
#
# # A more efficient alternative for *nix system users
# # Note: replace "cat" by "zcat" if files are gzipped
# haps.files = paste("example_chr",1:2,".haps",sep="")
# command <- "tail -n+3 example_chr1.sample | cut -d' ' -f1,2 > example.samples"
# system(command)
# for(i in 1:2){
#   command <- paste("cat",haps.files[i],"| cut -d' ' -f1-5 >> example.markers")
#   system(command)
#  command <- paste("cat",haps.files[i],"| cut -d' ' -f1-5 --complement >> example.phase")
#   system(command)
# }
```

---

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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
# # Exclude minor alleles and singletons
# hapstats <- ghap.hapstats(haplo, ncores = 1)
# haplo <- ghap.subsethaplo(haplo,ids=haplo$id, alleles = hapstats$TYPE %in% c("REGULAR","MAJOR"))
#
# # Compute Kinship matrix
# K <- ghap.kinship(haplo, ncores = 1)
#
#
# ### 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, batchsize = NULL,
             ncores = 1, verbose = TRUE)
```

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).
`batchsize`	A numeric value controlling the number of HapAlleles to be processed at a time (default = nalleles/10).
`ncores`	A numeric value specifying the number of cores to be used in parallel computing (default = 1).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[which(haplo$pop %in% c("TSI","CEU"))]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,haplo$allele.in)
# hapstats <- ghap.hapstats(haplo, ncores = 1)
# common <- hapstats$TYPE %in% c("REGULAR","MAJOR") &
#  hapstats$FREQ > 0.05 &
```

```
#  hapstats$FREQ < 0.95
# haplo <- ghap.subsethaplo(haplo,EUR.ids,common)
#
# # Compute relationship matrix
# K <- ghap.kinship(haplo, ncores = 1)
#
# # Quantitative trait with 50% heritability
# # Unbalanced repeated measurements (0 to 30)
# # Two major haplotypes accounting for 50% of the genetic variance
# myseed <- 123456789
# set.seed(myseed)
# major <- sample(which(haplo$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$data)
#
# # BLUP GWAS
# gebvs <- model$random$individual
# Kinv <- ghap.kinv(K)
# gwas.blup <- ghap.blup(gebvs = gebvs, haplo = haplo,
#                        ncores = 1, invcov = Kinv)
#
#
# ### RUN ###
#
# # Scoring
# score <- ghap.profile(score = gwas.blup, haplo = haplo, ncores = 1)
# plot(score$SCORE, model$random$individual); abline(0,1)
```

`ghap.pslice` *Get a slice of the phase object*

Description

This function parses the binary phased genotype matrix and returns the slice as an R matrix.

Usage

```
ghap.pslice(phase, ids, markers, index=FALSE, unphase=FALSE,
            lookup=NULL, ncores=1, verbose=TRUE)
```

Arguments

phase	A GHap.phase object.
ids	A character or numeric vector indicating individuals to parse.
markers	A character or numeric vector indicating markers to parse.
index	A logical value specfying if values provided for ids and markers are indices (see details).
unphase	A logical value specfying if phased genotypes should be retrieved as unphased allele counts.
lookup	A character vector containing the look up table to decode the binary phase file. Only meaningful for developers.
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).

Details

This function parses the file with suffix *.phaseb* and returns an R matrix with the requested list of markers and individuals. The argument index allows the user to specify markers and individuals either by name (index = FALSE) or by indices as stored in the GHap.phase object (index = TRUE). The lookup argument also allows the user to provide a vector containing the decoding scheme to translate bits into integers in R. However, this argument should only be used by developers.

Value

An R matrix with markers in rows and haplotypes in columns.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# ### RUN ###
#
```

```
# # Get 1000 random markers for CEU individuals
# ids <- unique(phase$id[which(phase$pop == "CEU")])
# markers <- sample(phase$marker, size = 1000, replace = FALSE)
# X <- ghap.pslice(phase = phase, ids = ids, markers = markers, index = FALSE)
#
# # Get the same data as before but unphased
# X <- ghap.pslice(phase = phase, ids = ids, markers = markers,
#                  index = FALSE, unphase = TRUE)
```

`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, ncores=1)
```

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.
`ncores`	A numeric value specifying the number of cores to be used in parallel computations (default = 1).

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
```

```
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
#
# ### RUN ###
#
# # Subset common haplotypes in Europeans
# EUR.ids <- haplo$id[which(haplo$pop %in% c("TSI","CEU"))]
# haplo <- ghap.subsethaplo(haplo,EUR.ids,haplo$allele.in)
# hapstats <- ghap.hapstats(haplo, ncores = 1)
# common <- hapstats$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, ncores = 1)
#
# # 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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
# # Generate blocks of 5 markers sliding 5 markers at a time
# blocks <- ghap.blockgen(phase, windowsize = 5, slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "human",
#                  binary = T, ncores = 1)
#
```

```
# # Load haplotype genotypes
# haplo <- ghap.loadhaplo(input.file = "human")
#
#
# ### 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
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Compress phase data
# ghap.compress(input.file = "human", out.file = "human")
#
# # Load phase data
# phase <- ghap.loadphase(input.file = "human")
#
#
# ### RUN ###
#
# # Subset data - markers with maf > 0.05
# maf <- ghap.freq(phase, type = "maf", ncores = 1)
# markers <- phase$marker[maf > 0.05]
# phase <- ghap.subsetphase(phase, unique(phase$id), markers)
```

---

ghap.vcf2phase *Convert VCF data into GHap phase*

Description

This function takes phased genotype data in the Variant Call Format (VCF) and converts them into the GHap phase format.

Usage

```
ghap.vcf2phase(input.files = NULL, vcf.files = NULL, sample.files = NULL,
               out.file, verbose = TRUE)
```

Arguments

	If all input files share the same prefix, the user can use the following shortcut options:
	Character vector with the list of prefixes for input files.
output.files	Character value for the output file name.
	The user can also opt to point to input files separately:
vcf.files	Character vector containing the list of VCF files.
sample.files	Character vector containing the list of SAMPLES files.
	To turn conversion progress-tracking on or off please use:
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

The Variant Call Format (VCF) - as described in https://github.com/samtools/hts-specs - is here manipulated to obtain the GHap phase format. Important: the function does not apply filters to the data, except for skipping multi-allelic variants. Should variants be filtered, the user is advised to pre-process the VCF files with third-party software (such as BCFTools). The FORMAT field should also follow the "GT:..." specification, with genotypes placed first in each sample column. Finally, all genotypes should be phased and take one of the following values: "0|0", "0|1", "1|0" or "1|1". Warning: this function is not optimized for large datasets.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

H. Li et al. The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics. 2009. 25:2078-2079.

H. Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. Bioinformatics. 2011. 27(21):2987-2993.

See Also

`ghap.compress`, `ghap.loadphase`, `ghap.fast2phase`, `ghap.oxford2phase`

Index