Package: GHap (via r-universe)

October 11, 2024

Type Package

Title Genome-Wide Haplotyping

Version 4.0.0.12

Date 2024-10-11

Author Yuri Tani Utsunomiya

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>). The source code for v3 was modified for improved performance and inclusion of new functionality, including analysis of unphased data, runs of homozygosity, sampling methods for virtual gamete mating, mixed model fitting and GWAS.

License GPL (>= 2)

Imports parallel (>= 3.4.4), Matrix (>= 1.2-16), methods (>= 3.4.4), pedigreemm (>= 0.3-3), sparseinv (>= 0.1.3), e1071 (>= 1.7-0.1), data.table (>= 1.12.6), stringi (>= 1.7.6), Rcpp (>= 1.0.6)

LinkingTo Rcpp

VignetteBuilder R.rsp

Suggests R.rsp

Encoding UTF-8

NeedsCompilation yes

Repository https://ytutsunomiya.r-universe.dev

RemoteUrl https://github.com/ytutsunomiya/ghap

RemoteRef HEAD

RemoteSha 65575819d4008f5d1359a74fab1abf563e94e1ff

Contents

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(object, 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

`object` 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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
#
# phase <- ghap.loadphase("example")
#
# # Unsupervised analysis
# prototypes <- ghap.anctrain(object = phase, K = 2)
# hapadmix <- ghap.anctest(object = phase,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
#
# ### RUN ###
#
# # Export crossbred data to PLINK binary
# cross <- unique(phase$id[which(phase$pop == "Cross")])
# phase <- ghap.subset(object = phase,
#                      ids = cross,
#                      variants = phase$marker)
# ghap.anc2plink(object = phase, ancsmooth = anctracks,
#                ancestry = "K1", outfile = "cross_K1")
```

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(object, ancsmooth, ids)
```

Arguments

object	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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
#
# phase <- ghap.loadphase("example")
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window,
#                         slide = step, unit = "marker")
#
# # Supervised analysis
# train <- unique(phase$id[which(phase$pop != "Cross")])
# prototypes <- ghap.anctrain(object = phase, train = train,
#                             method = "supervised")
# hapadmix <- ghap.anctest(object = phase,
#                          blocks = blocks,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
#
# ### RUN ###
#
# # Get per marker ancestry proportions for 'Pure1'
# pure1 <- unique(phase$id[which(phase$pop == "Pure1")])
# ancmark <- ghap.ancmark(object = phase,
#                         ancsmooth = anctracks,
#                         ids = pure1)
#
# # Plot 'Pure2' introgression into 'Pure1'
# ghap.manhattan(data = ancmark, chr = "CHR",
#                bp = "BP", y = "Pure2", type = "h")
```

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 = FALSE,
             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

Y.T. Utsunomiya et al. Unsupervised detection of ancestry tracks with the GHap R package. Methods in Ecology and Evolution. 2020. 11:1448–54.

See Also

`ghap.ancsmooth`, `ghap.ancmark`

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
#
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window,
#                         slide = step, unit = "marker")
#
# # BestK analysis
# bestK <- ghap.anctrain(object = phase, K = 5, tune = TRUE)
# plot(bestK$ssq, type = "b", xlab = "K",
#      ylab = "Within-cluster sum of squares")
#
# # Unsupervised analysis with best K
# prototypes <- ghap.anctrain(object = phase, K = 2)
# hapadmix <- ghap.anctest(object = phase,
#                          blocks = blocks,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
#
# # Supervised analysis
# train <- unique(phase$id[which(phase$pop != "Cross")])
# prototypes <- ghap.anctrain(object = phase, train = train,
#                             method = "supervised")
# hapadmix <- ghap.anctest(object = phase,
#                          blocks = blocks,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
```

`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(object, admix,
               ncores = 1, verbose = TRUE)
```

Arguments

`object` 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>

References

Y.T. Utsunomiya et al. Unsupervised detection of ancestry tracks with the GHap R package. Methods in Ecology and Evolution. 2020. 11:1448–54.

See Also

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

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
#
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window,
#                         slide = step, unit = "marker")
#
# # BestK analysis
# bestK <- ghap.anctrain(object = phase, K = 5, tune = TRUE)
# plot(bestK$ssq, type = "b", xlab = "K",
#      ylab = "Within-cluster sum of squares")
#
# # Unsupervised analysis with best K
# prototypes <- ghap.anctrain(object = phase, K = 2)
# hapadmix <- ghap.anctest(object = phase,
#                          blocks = blocks,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
```

```
#
# # Supervised analysis
# train <- unique(phase$id[which(phase$pop != "Cross")])
# prototypes <- ghap.anctrain(object = phase, train = train,
#                             method = "supervised")
# hapadmix <- ghap.anctest(object = phase,
#                          blocks = blocks,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
```

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(object, blocks, test = NULL, train = NULL,
            cost = 1, gamma = NULL, tune = FALSE,
            only.active.samples = TRUE, only.active.markers = TRUE,
            ncores = 1, verbose = TRUE)
```

Arguments

object	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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
#
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window,
#                         slide = step, unit = "marker")
#
# # Tune supervised analysis
# train <- unique(phase$id[which(phase$pop != "Cross")])
# ranblocks <- sample(x = 1:nrow(blocks), size = 5, replace = FALSE)
# tunesvm <- ghap.ancsvm(object = phase, blocks = blocks[ranblocks,],
#                        train = train, gamma = 1/window*c(0.1,1,10),
#                        tune = TRUE)
#
# # Supervised analysis with default parameters
# hapadmix <- ghap.ancsvm(object = phase, blocks = blocks,
#                         train = train)
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
```

---

ghap.anctest *Prediction of haplotype ancestry*

Description

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

Usage

```
ghap.anctest(object, blocks = NULL,
             prototypes, test = NULL,
             only.active.samples = TRUE,
             only.active.markers = TRUE,
             ncores = 1, verbose = TRUE)
```

Arguments

`object`	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). If no blocks are supplied, the function automatically builds blocks compatible with admixture up to 10 generations in the past based on intermarker distances. This has been chosen according to simulation results, where the use of haplotype blocks compatible with recent admixture (~10 generations) retained reasonable accuracy across most scenarios, regardless of the age of admixture.

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>

References

Y.T. Utsunomiya et al. Unsupervised detection of ancestry tracks with the GHap R package. Methods in Ecology and Evolution. 2020. 11:1448–54.

See Also

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

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
#
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window,
#                         slide = step, unit = "marker")
#
# # BestK analysis
# bestK <- ghap.anctrain(object = phase, K = 5, tune = TRUE)
# plot(bestK$ssq, type = "b", xlab = "K",
#      ylab = "Within-cluster sum of squares")
#
# # Unsupervised analysis with best K
# prototypes <- ghap.anctrain(object = phase, K = 2)
# hapadmix <- ghap.anctest(object = phase,
#                          blocks = blocks,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
#
# # Supervised analysis
# train <- unique(phase$id[which(phase$pop != "Cross")])
# prototypes <- ghap.anctrain(object = phase, train = train,
#                             method = "supervised")
# hapadmix <- ghap.anctest(object = phase,
#                          blocks = blocks,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
```

ghap.anctrain *Construction of prototype alleles*

Description

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

Usage

```
ghap.anctrain(object, 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:

object	A GHap.phase object.
train	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>

References

Y.T. Utsunomiya et al. Unsupervised detection of ancestry tracks with the GHap R package. Methods in Ecology and Evolution. 2020. 11:1448–54.

See Also

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

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
#
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window,
#                          slide = step, unit = "marker")
#
# # BestK analysis
# bestK <- ghap.anctrain(object = phase, K = 5, tune = TRUE)
# plot(bestK$ssq, type = "b", xlab = "K",
#      ylab = "Within-cluster sum of squares")
#
# # Unsupervised analysis with best K
# prototypes <- ghap.anctrain(object = phase, K = 2)
# hapadmix <- ghap.anctest(object = phase,
#                           blocks = blocks,
#                           prototypes = prototypes,
#                           test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
#
# # Supervised analysis
# train <- unique(phase$id[which(phase$pop != "Cross")])
# prototypes <- ghap.anctrain(object = phase, train = train,
#                              method = "supervised")
# hapadmix <- ghap.anctest(object = phase,
#                           blocks = blocks,
#                           prototypes = prototypes,
#                           test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
```

ghap.assoc *Genome-wide association analysis*

Description

This function performs phenotype-genotype association analysis based on mixed models.

Usage

```
ghap.assoc(object, formula, data,
           covmat, batchsize = NULL,
           ngamma = 100, nlambda = 1000,
           recalibrate = 0.01,
           only.active.variants=TRUE,
           tol = 1e-12, ncores=1,
           verbose=TRUE, ...)
```

Arguments

object A valid GHap object (phase, haplo or plink).

`formula` Formula describing the model. The synthax is consistent with lme4. The response is declared first, followed by the ~ operator. Predictors are then separated by + operators. Currently only random intercepts are supported, which are distinguished from fixed effects by the notation (1|x). If multiple random effects are specified, the first declared in the formula will be assumed to be the genetic (polygenic) effects.

`data` A dataframe containing the data.

`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. Inverse covariance matrices can also be provided, as long as argument invcov = TRUE is used.

`batchsize` A numeric value controlling the number of variants to be processed at a time (default = nalleles/10)

`ngamma` A numeric value for the number of variants to be used in the estimation of the gamma factor (default = 100). The grammar-gamma approximation is turned off if ngamma = 0.

`nlambda` A numeric value for the number of variants to be used in the estimation of the inflation factor (default = 1000).

`recalibrate` A numeric value for the proportion of top scoring variants to re-analyze without the grammar-gamma approximation (default = 0.01). Not relevant if ngamma = 0.

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

`tol` A numeric value specifying the scalar to add to the diagonal of the phenotypic (co)variance matrix if it is not inversible (default = 1e-12).

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

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

`...` Additional arguments to be passed to the `ghap.lmm` function.

Details

This function uses mixed models and the grammar-gamma approximation for fast genome-wide association analysis. Since mixed models are fit using the `ghap.lmm` function, the association analysis can be performed using more flexible models than those offered by alternative software, including the use of repeated measurements and other random effects apart from polygenic effects.

Value

The function returns a data frame with results from the genome-wide association analysis. If a GHap.haplo object is used, the first columns of the data frame will be:

`CHR` Chromosome name.

`BLOCK` Block alias.

`BP1` Block start position.

`BP2` Block end position.

For GHap.phase and GHap.plink objects, the first columns will be:

`CHR` Chromosome name.

`MARKER` Block start position.

`BP` Block end position.

The remaining columns of the data frame will be equal for any class of GHap objects:

`ALLELE` Identity of the counted (A1 or haplotype) allele.

`FREQ` Frequency of the allele in the sample of genotypes.

`N.PHENO` Number of non-missing phenotypic observations.

`N.GENO` Number of non-missing genotypic observations.

`BETA` Estimated allele effect.

`SE` Standard error for the allele effect.

`CHISQ.EXP` Expected values for the test statistics.

`CHISQ.OBS` Observed value for the test statistics.

`CHISQ.GC` Test statistics scaled by the inflation factor (Genomic Control). Inflation is computed through regression of observed quantiles onto expected quantiles. In order to avoid overestimation by variants rejecting the null hypothesis, a random sample of variants (with size controled via the nlambda argument) is taken within three standard deviations from the mean of the distribution of test statistics.

`LOGP` log10(1/P) or -log10(P) for the allele substitution effect.

`LOGP.GC` log10(1/P) or -log10(P) for the allele substitution effect (scaled by the inflation factor).

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.

L. Jiang et al. A resource-efficient tool for mixed model association analysis of large-scale data. Nat. Genet. 2019. 51:1749-1755.

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 plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load phenotype data
# df <- read.table(file = "example.phenotypes", header=T)
#
# ### RUN ###
#
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute genomic relationship matrix
# # Induce sparsity to help with matrix inversion
# K <- ghap.kinship(plink, sparsity = 0.01)
#
# # Perform GWAS on repeated measures
# # Use grammar-gama approximation
# # Recalibrate top 1 percent variants
# df$rep <- df$id
# gwas1 <- ghap.assoc(object = plink,
#                     formula = pheno ~ 1 + (1|id) + (1|rep),
#                     data = df,
#                     covmat = list(id = K, rep = NULL),
#                     ngamma = 100, nlambda = 1000, recalibrate = 0.01)
# ghap.manhattan(data = gwas1, chr = "CHR", bp = "BP", y = "LOGP")
```

```
#
# # GWAS with no approximaion (slow)
# gwas2 <- ghap.assoc(object = plink,
#                     formula = pheno ~ 1 + (1|id) + (1|rep),
#                     data = df,
#                     covmat = list(id = K, rep = NULL),
#                     ngamma = 0, nlambda = 1000)
# ghap.manhattan(data = gwas2, chr = "CHR", bp = "BP", y = "LOGP")
#
# # Correlation between methods
# cor(gwas1$LOGP, gwas2$LOGP)
# plot(gwas1$LOGP, gwas2$LOGP); abline(0,1)
```

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(object, windowsize = 10, slide = 5,
              unit = "marker", nsnp = 2)
```

Arguments

object	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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load data
# phase <- ghap.loadphase("example")
#
# ### 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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load data
# phase <- ghap.loadphase("example")
#
# # Generate blocks of 5 markers
# blocks <- ghap.blockgen(phase, windowsize = 5,
#                         slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "example",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes using prefix
# haplo <- ghap.loadhaplo("example")
#
# # Subset
# ids <- which(haplo$pop == "Pure1")
# haplo <- ghap.subset(haplo, ids = ids,
#                      variants = haplo$allele.in,
#                      index = TRUE)
```

```
#
# # Compute haplotype statistics
# hapstats <- ghap.hapstats(haplo)
#
# ### RUN ###
#
# # Compute block statistics
# blockstats <- ghap.blockstats(hapstats)
```

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, mode = 1,
              ncores = 1, verbose = TRUE)
```

Arguments

If all input files share the same prefix, the user can use the following shortcut options:

input.file	Prefix for input files.
out.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, choose file format or control parallelization of the task please use:

mode	A numeric value indicating the input file format and consequently the output binary file format. Mode 1 starts with byte 00000001 and informs that the matrix is organized as variants x individuals. Mode 2 starts with byte 00000010 and indicates individuals x variants. For backward compatibility, mode 0 has no leading byte and is also organized as variants x individuals.
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 supported input format is composed of three files with suffix:

- **.samples**: space-delimited file without header containing two mandatory 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. This file may further contain three additional columns, which are optional: Sire, Dam and Sex (with code 1 = M and 2 = F). Values "0" and "NA" in these additional columns are treated as missing values.
- **.markers**: space-delimited file without header containing five mandatory columns: Chromosome, Marker, Position (in bp), Reference Allele (A0) and Alternative Allele (A1). Markers should be sorted by chromosome and position. Repeated positions are tolerated, but the user is warned of their presence in the data. Optionally, the user may provide a file containing an additional column with genetic positions (in cM), which has to be placed between the base pair position and the reference allele columns.
- **.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>, Adam Taiti Harth Utsunomiya <adamtaiti@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "raw",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
#
# # Compress phase data using prefix
# ghap.compress(input.file = "example",
#               out.file = "example")
#
# # Compress phase data using file names
# ghap.compress(samples.file = "example.samples",
#               markers.file = "example.markers",
#               phase.file = "example.phase",
#               out.file = "example")
```

ghap.exfiles *Example files*

Description

This function retrieves the list of example files available.

Usage

```
ghap.exfiles()
```

Details

This function requires internet connection. It returns a data table containing the list of example files in our github repository (https://github.com/ytutsunomiya/GHap). To get any of those files, please use ghap.makefile.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# # See list of example files
# exlist <- ghap.exfiles()
# View(exlist)
```

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, fam.file = NULL,
                out.file = NULL, overwrite = FALSE,
                ncores = 1, verbose = TRUE)
```

Arguments

If all input files share the same prefix, the user can use the following shortcut options:

`input.files` Character vector with the list of prefixes for input files.

`out.file` 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.

`fam.file` Name of the file containing the population and individual ids.

The function avoids overwritting existing files by default. To change this behavior, please use:

`overwrite` A logical value controling if existing files with the same name as the selected output should be overwritten (default = FALSE).

To turn conversion progress-tracking on or off or set the number of cores please use:

`ncores` A numeric value specfying the number of cores (default = 1).

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

Details

Currently this function handles _switch.out files from fastPHASE v1.4.0 or later. The map files should contain the following 5 space-delimited columns: chromosome, marker, position, allele 0 and allele 1 (no header). The fam file can have an arbitrary number of columns (with no header), but by default only the first two are read and should be space-delimited with the following data: population and individual name.

Author(s)

Mario Barbato <mario.barbato@unicatt.it>

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(object, type = "maf",
          batchsize = NULL,
          only.active.samples = TRUE,
          only.active.variants = TRUE,
          verbose = TRUE)
```

Arguments

`object`	A GHap object of type phase or plink.
`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').
`batchsize`	A numeric value controlling the number of variants to be processed at a time (default = nalleles/10).
`only.active.samples`	A logical value specifying whether only active samples should be used for calculations (default = TRUE).
`only.active.variants`	A logical value specifying whether only active variants should be included in the output (default = TRUE).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Value

The function outputs 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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
#
# # Calculate allele frequency for phase data
# phase <- ghap.loadphase("example")
# q <- ghap.freq(phase, type = 'A0')
# p <- ghap.freq(phase, type = 'A1')
# maf <- ghap.freq(phase, type = 'maf')
#
# # Calculate allele frequency for plink data
# plink <- ghap.loadplink("example")
# q <- ghap.freq(plink, type = 'A0')
# p <- ghap.freq(plink, type = 'A1')
# maf <- ghap.freq(plink, type = 'maf')
```

ghap.froh *Calculation of genomic inbreeding (FROH)*

Description

Given runs of homozygosity (ROH) obtained with the ghap.roh function, this function computes the proportion of the genome covered by ROHs (FROH) of certain lengths.

Usage

```
ghap.froh(object, roh, rohsizes = c(1, 2, 4, 8, 16),
          only.active.markers = TRUE, ncores = 1)
```

Arguments

object A valid GHap object (phase or plink).

roh A data frame containing runs of homozygosity, such as supplied by the ghap.roh function.

`rohsizes` A numeric vector providing the minimum ROH length (in Mbp) to use in the calculation of ROH (default is 1, 2, 4, 8 and 16).

`only.active.markers`
A logical value specifying whether only active markers should be used in the calculation of genome size (default = TRUE).

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

Details

This function takes runs of homozygosity obtained with `ghap.roh` and returns estimates of genomic inbreeding (FROH). The user can specify the minimum ROH length considered in the calculation using the rohsize argument. A vector of values will cause the function to add an extra column for each specified ROH size. Since the average size (measured in Morgans) of identical-by-descent segments after g generations of the inbreeding event is 1/2g, the default lengths 1, 2, 4, 8 adn 16 are proxies for inbreeding that occurred 50, 25, 13 6 and 3 generations in the past, respectively (assuming an average recombination rate of 1 Mbp ~ cM).

Value

The function returns a dataframe the following columns:

`POP` Original population label.

`ID` Individual name.

`FROH...` A number of columns giving FROH calculated over runs of length greater than each of the sizes informed by the rohsize argument. Default values will return FROH1, FROH2, FROH4, FROH8 and FROH16.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

See Also

`ghap.roh`

Examples

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

```
#
# # Subset pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # ROH via the 'naive' method
# roh1 <- ghap.roh(plink, method = "naive")
# froh1 <- ghap.froh(plink, roh1)
#
# # ROH via the 'hmm' method
# freq <- ghap.freq(plink, type = 'A1')
# inbcoef <- froh1$FROH1; names(inbcoef) <- froh1$ID
# roh2 <- ghap.roh(plink, method = "hmm", freq = freq,
#                  inbcoef = inbcoef)
# froh2 <- ghap.froh(plink, roh2)
#
# # Method 'hmm' using Fhat3 as starting values
# inbcoef <- ibc$Fhat3; names(inbcoef) <- ibc$ID
# inbcoef[which(inbcoef < 0)] <- 0.01
# roh3 <- ghap.roh(plink, method = "hmm", freq = freq,
#                   inbcoef = inbcoef)
# froh3 <- ghap.froh(plink, roh3)
```

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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load data
# phase <- ghap.loadphase("example")
#
# # Generate blocks of 5 markers
# blocks <- ghap.blockgen(phase, windowsize = 5,
#                         slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "example",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes using prefix
# haplo <- ghap.loadhaplo("example")
#
# ### RUN ###
#
# # Compute block statistics for population 1
# ids <- which(haplo$pop == "Pure1")
# haplo <- ghap.subset(haplo, ids = ids,
#                      variants = haplo$allele.in,
#                      index = TRUE)
# hapstats1 <- ghap.hapstats(haplo)
# blockstats1 <- ghap.blockstats(hapstats1)
#
# # Compute block statistics for population 2
# ids <- which(haplo$pop == "Pure2")
# haplo <- ghap.subset(haplo, ids = ids,
#                      variants = haplo$allele.in,
#                      index = TRUE)
# hapstats2 <- ghap.hapstats(haplo)
# blockstats2 <- ghap.blockstats(hapstats2)
#
# # Compute block statistics for combined populations
# ids <- which(haplo$pop %in% c("Pure1","Pure2"))
# haplo <- ghap.subset(haplo, ids = ids,
#                      variants = haplo$allele.in,
#                      index = TRUE)
# hapstats12 <- ghap.hapstats(haplo)
# blockstats12 <- ghap.blockstats(hapstats12)
#
# # Compute FST
# fst <- ghap.fst(blockstats1, blockstats2, blockstats12)
# ghap.manhattan(data = fst, chr = "CHR", bp = "BP1",
#                y = "FST", type = "h")
```

ghap.getHinv *Compute the inverse of H*

Description

This function combines an additive genomic relationship matrix with pedigree data to form the inverse of the H relationship matrix used in single-step GBLUP.

Usage

```
ghap.getHinv(K, ped, include = NULL, depth = 3,
              alpha = 0.95, verbose=TRUE)
```

Arguments

K An additive genomic relationship matrix, as provided for example by type=3 in ghap.kinship.

ped A dataframe with columns "id", "sire" and "dam" containing pedigree data.

include An optional vector of ids to be forced into the output (default = NULL). See details.

depth A numeric value specifying the pedigree depth in number of generations to be used (default = 3). See details.

alpha A numeric value between 0 and 1 specifying the weight of the genomic relationship matrix in the blend alpha*K + (1-alpha)*K. Default = 0.95.

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

Details

The pedigree is pruned to include only the number of generations specified by argument "depth". This prunning starts by seeding the genealogy with all individuals included in the genomic relationship matrix plus all individuals listed in the "include" argument. Then, the genealogy is increased by advancing one generation back at a time up to "depth". For example, if depth = 3, the genealogical tree will include all individuals listed in K and "include", plus their parents (depth = 1), grandparents (depth = 2) and great-grandparents (depth = 3). After the pedigree has been pruned, pedigree and genomic relationships are blended to form the inverse of H.

Value

A matrix consisting of the inverse of H.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

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 plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load phenotype and pedigree data
# df <- read.table(file = "example.phenotypes", header=T)
# ped <- read.table(file = "example.pedigree", header=T)
#
# ### RUN ###
#
# # This analysis emulates a scenario of
# # 100 individuals with genotypes and phenotypes
# # 200 individuals with only phenotypes
# # 400 individuals from pedigree with no data
#
# # Subset 100 individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# pure1 <- sample(x = pure1, size = 100)
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute genomic relationship matrix
# # Induce sparsity to help with matrix inversion
# K <- ghap.kinship(plink, sparsity = 0.01)
#
# # Exclude pedigree records with missing sire
# ped <- ped[which(is.na(ped$sire) == F),]
#
# # Make inverse of blended pedigree/genomic matrix
# ids <- unique(c(ped$id, ped$sire, ped$dam, colnames(K)))
# Hinv <- ghap.getHinv(K = K, ped = ped[,-1], include = ids)
#
# # Run single-step GBLUP
# df$rep <- df$id
# model <- ghap.lmm(formula = pheno ~ 1 + (1|id) + (1|rep),
#                   data = df,
#                   covmat = list(id = Hinv, rep = NULL),
#                   invcov = T)
```

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(object, outfile)
```

Arguments

`object`	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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
#
# # Load data
# phase <- ghap.loadphase("example")
#
# # Generate blocks of 5 markers
# blocks <- ghap.blockgen(phase, windowsize = 5,
#                         slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "example",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes using prefix
# haplo <- ghap.loadhaplo("example")
#
# ### RUN ###
#
# # Convert to plink
# ghap.hap2plink(haplo, outfile = "example")
```

ghap.haplotyping *Haplotype genotypes*

Description

Generate matrix of HapGenotypes for user-defined blocks.

Usage

```
ghap.haplotyping(object, 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

`object`	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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load data
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Generate blocks
# blocks <- ghap.blockgen(phase, windowsize = 5,
#                         slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase, blocks = blocks,
#                  outfile = "example",
#                  binary = T, ncores = 1)
```

ghap.hapstats *Haplotype allele statistics*

Description

Summary statistics for HapAlleles.

Usage

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

Arguments

`object` 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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
```

```
#
# # Load data
# phase <- ghap.loadphase("example")
#
# # Generate blocks of 5 markers
# blocks <- ghap.blockgen(phase, windowsize = 5,
#                         slide = 5, unit = "marker")
#
# # Haplotyping
# ghap.haplotyping(phase = phase, blocks = blocks, outfile = "example",
#                  binary = T, ncores = 1)
#
# # Load haplotype genotypes using prefix
# haplo <- ghap.loadhaplo("example")
#
# ### RUN ###
#
# # Subset
# ids <- which(haplo$pop == "Pure1")
# haplo <- ghap.subset(haplo, ids = ids,
#                      variants = haplo$allele.in,
#                      index = TRUE)
#
# # Compute haplotype statistics
# hapstats <- ghap.hapstats(haplo)
```

ghap.ibd *Estimation of IBD sharing*

Description

This function estimates the same IBD statistics computed by plink's 'genome' option.

Usage

```
ghap.ibd(object, pairlist, freq, mafcut=0.05,
         refsize=10000, batchsize=NULL,
         ncores=1, verbose=TRUE)
```

Arguments

`object` A valid GHap object (phase or plink).

`pairlist` A dataframe containing columns ID1 and ID2 specifying the pairs of individual ids to compare.

`freq` A named numeric vector with (A1) allele frequencies computed in a reference sample.

`mafcut` A numeric value specifying the minor allele frequency threshold for IBD calculations (default = 0.05).

`refsize` A numeric value representing the reference sample size used in allele frequncy calculations. If not specified, a large reference sample is assumed (default = 10000)

`batchsize` A numeric value controlling the number of variants 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 implements plink's method-of-moments for IBD estimation. Although not as efficient as plink's implementation, our function allows the user to restrict calculations to specific individual pairs, as well as ground all computations on allele frequencies obtained from a reference population. This is useful for routine pedigree confirmation, since a smaller set of indviduals and comparisons are typically targeted in these situations. The original –genome flag in plink not only performs all possible comparisons given a set of individuals, but also estimates allele frequencies on-the-fly, which may be unreliable if the number of individuals is small. We still recommend using plink for large problems, such as all pairwise comparisons from thousands of individuals, because it is more efficient. Nevertheless, we offer a more convenient alternative for the validation of smaller pedigrees in routine analyses where a lookup table of allele frequencies is available and maintained from a large reference population.

Value

A dataframe with columns: POP1 = Population of individual 1
ID1 = Name of individual 1
POP2 = Population of individual 2
ID2 = Name of individual 2
IBS0 = Variant sites where ID1 and ID2 share no identical alleles
IBS1 = Variant sites where ID1 and ID2 share 1 identical allele
IBS2 = Variant sites where ID1 and ID2 share 2 identical alleles
PERC = IBS2/(IBS0+IBS1+IBS2) [proportion of identical genotypes]
DST = (IBS2 + 0.5*IBS1)/(IBS0+IBS1+IBS2) [proportion of shared alleles]
Z0 = Proportion of the genome where ID1 and ID2 share no alleles IBD
Z1 = Proportion of the genome where ID1 and ID2 share 1 allele IBD
Z2 = Proportion of the genome where ID1 and ID2 share 2 alleles IBD
PI_HAT = Z2 + 0.5*Z1 (proportion of IBD alleles shared between ID1 and ID2)

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.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load pedigree data
# ped <- read.table(file = "example.pedigree", header=T)
#
# ### RUN ###
#
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute A1 allele frequencies
# p <- ghap.freq(plink, type = "A1")
#
# # Compute IBD statistics for individual 1
# pairlist <- data.frame(ID1 = pure1[1], ID2 = pure1[-1])
# ibd <- ghap.ibd(object = plink, pairlist = pairlist, freq = p,
#                 refsize = length(pure1))
#
# # Predict relationships for individual 1
# # 1 = parent-offspring
# # 3 = other types of relationship
# # 4 = unrelated
# rel <- ghap.relfind(ibdpairs = ibd)
# table(rel$REL)
#
# # Confirm with pedigree
# toprel <- rel$ID2[which(rel$REL == 1)]
# ped[which(ped$id %in% toprel & ped$dam == pure1[1]),]
```

ghap.inbcoef *Compute measures of inbreeding*

Description

This function computes genomic measures of inbreeding.

Usage

```
ghap.inbcoef(object, freq, batchsize=NULL,
             only.active.samples=TRUE,
             only.active.variants=TRUE,
             ncores=1, verbose=TRUE)
```

Arguments

`object` A valid GHap object (phase, haplo or plink).

`freq` A named numeric vector providing allele frequencies.

`batchsize` A numeric value controlling the number of variants 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.variants` A logical value specifying whether only active variants 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

The inbreeding measures are computed as k + s*sum(f)/d, where k is a constant, s is a sign shifting scalar, f is a per-variant function and d is a scaling denominator. Four different measures of inbreeding are currently available:

Type = 1 (based on genomic relationship)
k = -1
s = 1
f = ((m - 2*p)^2)/(2*p*(1-p))
d = n

Type = 2 (excess homozygosity)
k = 1
s = -1

f = m*(2-m)/(2*p*(1-p))
d = n

Type = 3 (correlation between uniting gametes)
k = 0
s = 1
f = (m^2 - (1+2*p) + 2*p^2)/(2*p*(1-p))
d = n

Type = 4 (method-of-moments)
k = 1
s = -1
f = length(het)
d = sum(2*p*(1-p))

In the expressions above, m is the genotype coded as 0, 1 or 2 copies of A1, p is the frequency of A1, n is the number of variants (only non-monomorphic ones are considered), and het is the number of heterozygous genotypes.

Value

The function returns a dataframe containing population name, id and inbreeding measures for each individual.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

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 plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# ### RUN ###
#
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute A1 allele frequencies
# p <- ghap.freq(plink, type = "A1")
#
# # Compute inbreeding coefficients
# ibc <- ghap.inbcoef(object = plink, freq = p)
```

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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
#
# phase <- ghap.loadphase("example")
#
# # Calculate marker density
# mrkdist <- diff(phase$bp)
# mrkdist <- mrkdist[which(mrkdist > 0)]
# density <- mean(mrkdist)
#
# # Generate blocks for admixture events up to g = 10 generations in the past
# # Assuming mean block size in Morgans of 1/(2*g)
# # Approximating 1 Morgan ~ 100 Mbp
# g <- 10
# window <- (100e+6)/(2*g)
# window <- ceiling(window/density)
# step <- ceiling(window/4)
# blocks <- ghap.blockgen(phase, windowsize = window,
#                         slide = step, unit = "marker")
#
# # Supervised analysis
# train <- unique(phase$id[which(phase$pop != "Cross")])
# prototypes <- ghap.anctrain(object = phase, train = train,
#                             method = "supervised")
# hapadmix <- ghap.anctest(object = phase,
#                          blocks = blocks,
#                          prototypes = prototypes,
#                          test = unique(phase$id))
# anctracks <- ghap.ancsmooth(object = phase, admix = hapadmix)
# ghap.ancplot(ancsmooth = anctracks)
```

```
#
# ### RUN ###
#
# # Plot karyoplot
# pure1 <- unique(phase$id[which(phase$pop == "Pure1")])
# pure2 <- unique(phase$id[which(phase$pop == "Pure2")])
# cross <- unique(phase$id[which(phase$pop == "Cross")])
# ghap.karyoplot(ancsmooth = anctracks, ids = pure1[1],
#                chr.line = 11, plot.line = 50, las=1, chr=NULL)
# ghap.karyoplot(ancsmooth = anctracks, ids = pure2[1],
#                chr.line = 11, plot.line = 50, las=1, chr=NULL)
# ghap.karyoplot(ancsmooth = anctracks, ids = cross[1],
#                chr.line = 11, plot.line = 50, las=1, chr=NULL)
```

`ghap.kinship` *Relationship matrix based on genomic data*

Description

This function computes marker-based and HapAllele-based relationship matrices.

Usage

```
ghap.kinship(object, weights = NULL,
             sparsity = NULL, type = 1,
             batchsize = NULL, freq = NULL,
             idrow = NULL, idcol = NULL,
             only.active.samples = TRUE,
             only.active.variants = TRUE,
             ncores = 1, verbose = TRUE)
```

Arguments

`object` A valid GHap object (phase, haplo or plink).

`weights` A numeric vector providing variant-specific weights.

`sparsity` A numeric value specifying a relationship cut-off (default = NULL). All relationships below the specified cut-off will be set to zero, inducing sparsity into the relationship matrix.

`type` A numeric value indicating the type of relationship matrix (see details).

`batchsize` A numeric value controlling the number of variants to be processed at a time (default = nalleles/10).

`freq` A named numeric vector with (A1) allele frequencies computed in a reference sample. If not provided, frequencies will be computed using the active individuals.

`idrow` A character vector providing ids of individuals to be included in rows (default = NULL, see details).

`idcol` A character vector providing ids of individuals to be included in columns (default = NULL, see details).

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

`only.active.variants` A logical value specifying whether only active variants 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 *n* x *m* matrix of genotypes, where *n* is the number of individuals and *m* is the number of variants (i.e, markers or HapAlleles). Prior to computation, genotypes in matrix $\mathbf{M}$ are transformed according to the desired relationship type. After that transformation, the relationship matrix is computed as:

$$\mathbf{K} = q^{-1}\mathbf{MDM'}$$

where $\mathbf{D} = diag(d_i)$, d_i is the weight of variant *i* (default $d_i = 1$), and q is a scaling factor. The argument type controls the genotype transformation and the scaling factor, and includes the following option:
Type = 1 (General additive 1)
Genotype transformation: m - mean(m)
Scaling factor: $tr(\mathbf{MDM'})^{-1}n$

Type = 2 (General additive 2)
Genotype transformation: (m - mean(m))/sd(m)
Scaling factor: m

Type = 3 (VanRaden, 2008)
Genotype transformation: m - 2*p[j]
Scaling factor: 2*sum(p*(1-p))

Type = 4 (GCTA)
Genotype transformation: (m - 2*p[j])/sqrt(2*p[j]*(1-p[j]))
Scaling factor: m

Type = 5 (Dominance 1)
Genotype transformation:
0 = -2*p[j]^2
1 = 2*p[j]*(1-p[j])
2 = -2*(1-p[j])^2
Scaling factor: 4*sum(p^2*(1-p)^2).

Type = 6 (Dominance 2)
Genotype transformation:
0 = -2*p[j]^2

1 = 2*p[j]*(1-p[j])
2 = -2*(1-p[j])^2
Scaling factor:
$tr(\mathbf{MDM}')^{-1}n.$

The idrow and idcol arguments are intended for faster calculations when the user wishes to obtain a non-symmetric relationship matrix with a set of individuals in rows and a different set of individuals in columns. For example, training x testing samples, males x females, etc. Currently only supported if freq != NULL and for type 3 matrices.

Value

The function returns a *n* x *n* relationship matrix, where *n* is the number of individuals.

Author(s)

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

References

J. F. Taylor. Implementation and accuracy of genomic selection. Aquaculture 2014. 420, S8-S14. P. M. VanRaden. Efficient methods to compute genomic predictions. J. Dairy. Sci. 2008. 91:4414-4423. J. Yang et al. GCTA: A Tool for Genome-wide Complex Trait Analysis. Am. J. Hum. Genet. 2011. 88:76–82. Z. G. Vitezica et al. On the additive and dominant variance and covariance of individuals within the genomic selection scope. Genetics. 2013. 195:1223-1230.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load phenotype data
# df <- read.table(file = "example.phenotypes", header=T)
#
# ### RUN ###
#
# # Subset pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Compute different types of relationship matrices
# K1 <- ghap.kinship(plink, type = 1) # General additive 1
# K2 <- ghap.kinship(plink, type = 2) # General additive 2
# K3 <- ghap.kinship(plink, type = 3) # VanRaden 2008
# K4 <- ghap.kinship(plink, type = 4) # GCTA GRM
# K5 <- ghap.kinship(plink, type = 5) # Dominance 1
# K6 <- ghap.kinship(plink, type = 6) # Dominance 2
```

`ghap.lmm` *Linear mixed model*

Description

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

Usage

```
ghap.lmm(formula, data, covmat = NULL,
         weights = NULL, vcp.initial = NULL,
         vcp.estimate = TRUE, vcp.conv = 1e-12,
         errors = TRUE, invcov = FALSE,
         em.reml = 10, tol = 1e-12, extras = NULL,
         verbose = TRUE)
```

Arguments

`formula` Formula describing the model. The synthax is consistent with lme4. The response is declared first, followed by the ~ operator. Predictors are then separated by + operators. Currently only random intercepts are supported, which are distinguished from fixed effects by the notation (1|x).

`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. Inverse covariance matrices can also be provided, as long as argument invcov = TRUE is 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.

`vcp.initial` A list of initial values for variance components. If not provided, the sample variance will be equally divided across the variance components.

`vcp.estimate` A logical value specifying whether variance components should be estimated (default = TRUE). If FALSE, values passed to vcp.initial will be regarded as known variances.

`vcp.conv` A numeric value specifying the convergence criterion for variance components (default = 1e-12).

`errors` A logical value specifying whether standard errors should be computed (default = TRUE).

`invcov` A logical value specifying whether the provided covariance matrices are already inverted (default = FALSE).

`em.reml` A numeric value specifying the number of EM-REML iterations to carry out before switching to AI-REML (default = 10).

`tol` A numeric value specifying the scalar to add to the diagonal of the left hand side of mixed models equations if it is not inversible (default = 1e-12).

`extras` A character vector indicating extra output to be included in the results list. Currently supported extras are the full inverse of the coefficient matrix ("LHSi") and the phenotypic (co)variance matrix ("V").

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

Details

The function fits mixed models using a combination of EM-REML, AI-REML and PCG. Random regression and multivariate analyses are currently not supported.

Value

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

`fixed` A dataframe containing estimates, standard errors, t values and p-values of fixed effects.

`random` A list of dataframes, one for each group of random effects, containing estimates, standard errors and accuracy of random effects.

`fitted` A dataframe with fitted values using all effects, only fixed effects and only random effects.

`residuals` A dataframe with residuals from subtracting fitted values using all effects, only fixed effects and only random effects.

`vcp` A dataframe with estimates and standard errors of variance components.

`extras` A list of extra outputs requested by the user (see arguments for possible extras).

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

J. Jensen et al. Residual maximum likelihood estimation of (Co)variance components in multivariate mixed linear models using average information. J. Ind. Soc. Ag. Statistics 1997. 49, 215-236.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load phenotype and pedigree data
# df <- read.table(file = "example.phenotypes", header=T)
#
# ### RUN ###
#
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute genomic relationship matrix
# # Induce sparsity to help with matrix inversion
# K <- ghap.kinship(plink, sparsity = 0.01)
#
# # Fit mixed model with variance components estimation
# df$rep <- df$id
# model1 <- ghap.lmm(formula = pheno ~ 1 + (1|id) + (1|rep),
#                    data = df,
#                    covmat = list(id = K, rep = NULL))
#
# # Fit mixed model with fixed variance components
# df$rep <- df$id
# model2 <- ghap.lmm(formula = pheno ~ 1 + (1|id) + (1|rep),
#                    data = df,
#                    covmat = list(id = K, rep = NULL),
#                    vcp.initial = list(id = 0.4, rep = 0.2, Residual = 0.4),
#                    vcp.estimate = F)
```

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 phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load data
# phase <- ghap.loadphase("example")
#
# # 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 = "example",
#                  binary = T, ncores = 1)
#
# ### RUN ###
#
# # Load haplotype genotypes using prefix
# haplo <- ghap.loadhaplo("example")
#
# # Load haplotype genotypes using file names
# haplo <- ghap.loadhaplo(hapsamples.file = "example.hapsamples",
#                         hapalleles.file = "example.hapalleles",
#                         hapgenotypesb.file = "example.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,
               ncores = 1, verbose = TRUE)
```

Arguments

If all input files share the same prefix, the user can use the following shortcut option:

`input.file`	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, or use multiple cores, please use:

`ncores`	A numerical value specfying the number of cores to use while loading the input files (default = 1).
`verbose`	A logical value specfying whether log messages should be printed (default = TRUE).

Value

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

`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.

`sire` A character vector indicating sire names, as provided in the third column of the sample file (optional).

`dam` A character vector indicating dam names, as provided in the fourth column of the sample file (optional).

`sex` A character vector indicating individual sex, as provided in the fifth column of the sample file (optional). Codes are converted as follows: 0 = NA, 1 = Male and 2 = Female.

`chr` A character vector indicating chromosome identity for each marker.

`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.

`cm` A numeric vector with genetic positions for markers. This information is obtained from the third column of the marker map file if it contains 6 columns. Otherwise, if the map file contains only 5 columns, genetic positions are considered absent and approximated from physical positions (in this case assumed to be the third column) as 1 Mb ~ 1 cM.

`bp` A numeric vector with marker positions. This information is obtained from the third column of the marker map file if it contains 5 columns, or from the fourth column if it contains 6 columns.

`A0` A character vector with reference alleles. This information is obtained from the fourth column of the marker map file in case it contains 5 columns, or from the fifth column if it contains 6 columns.

`A1` A character vector with alternative alleles. This information is obtained from the last column of the marker map file.

`phase` A character value giving the pathway to the binary phased genotype matrix.

`mode` A numeric value indicating the binary file format. Mode 1 starts with byte 00000001 and informs that the matrix is organized as variants x individuals. Mode 2 starts with byte 00000010 and indicates individuals x variants. For backward compatibility, mode 0 has no leading byte and is also organized as variants x individuals.

Author(s)

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

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
```

```
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
#
# # Load data using prefix
# phase <- ghap.loadphase(input.file = "example")
#
# # Load data using file names
# phase <- ghap.loadphase(samples.file = "example.samples",
#                         markers.file = "example.markers",
#                         phaseb.file = "example.phaseb")
```

ghap.loadplink *Load binary PLINK data*

Description

This function loads binary PLINK files (bed/bim/fam) and converts them into a native GHap.plink object.

Usage

```
ghap.loadplink(input.file = NULL, bed.file = NULL,
               bim.file = NULL, fam.file = NULL,
               ncores = 1, verbose = TRUE)
```

Arguments

If all input files share the same prefix, the user can use the following shortcut option:

input.file	Prefix for input files.

For backward compatibility, the user can still point to input files separately:

bed.file	The binary genotype matrix (in SNP-major format).
bim.file	Variant map file.
fam.file	Pedigree (family) file.

To turn loading progress-tracking on or off, or engage multiple cores, please use:

ncores	A numerical value specfying the number of cores to use while loading the input files (default = 1).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Value

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

`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 genotypes to populations. This information is obtained from the FID (1st) column in the fam file.

`id` A character vector mapping genotypes to samples. This information is obtained from the IID (2nd) column in the fam file.

`id.in` A logical vector indicating active chromosome alleles. By default, all chromosomes are set to TRUE.

`sire` A character vector indicating sire names, as provided in the SID (3rd) column of the fam file.

`dam` A character vector indicating dam names, as provided in the DID (4th) column of the fam file.

`sex` A character vector indicating individual sex, as provided in the SEX (5th) column of the fam file. Codes are converted as follows: 0 = NA, 1 = Male and 2 = Female.

`chr` A character vector indicating chromosome identity for each marker.

`marker` A character vector containing marker names.

`marker.in` A logical vector indicating active markers. By default, all markers are set to TRUE.

`cm` A numeric vector with genetic positions for markers. This information is obtained from the third column of the bim file. If genetic positions are absent (coded as "0"), they are approximated from physical positions assuming 1 Mb ~ 1 cM.

`bp` A numeric vector with physical positions for markers.

`A0` A character vector with reference alleles. For convenience, this information is obtained from the 6th column of the bim file. If "–keep-allele-order" is not used while generating the PLINK binary file, A0 will correspond to the major allele.

`A1` A character vector with alternative alleles. As for A0, if "–keep-allele-order" is not used A1 will correspond to the minor allele.

`plink` A character value giving the pathway to the binary genotype matrix.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
#
# # Load data using prefix
# plink <- ghap.loadplink("example")
#
# # Load data using file names
# plink <- ghap.loadplink(bed.file = "example.bed",
#                         bim.file = "example.bim",
#                         fam.file = "example.fam")
```

---

ghap.makefile *Create example input files*

Description

Create example files to test the package.

Usage

```
ghap.makefile(dataset = "example",
              format = "phase",
              verbose = TRUE)
```

Arguments

`dataset` A character value specfying the name of the dataset.

`format` A character value specfying the format of the dataset.

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

Details

This function downloads example files to the R temporary directory (requires internet connection). The default dataset comprises the following group of files:

example.phaseb
example.markers

example.samples

For details about the format of these files, see ghap.compress. The dataset was simulated using the QMSim v1.10 software and contains 450 individuals genotyped for 15,000 markers. These markers were randomly distributed along 10 chromosomes of 100 Mbp each (i.e., 1,500 markers per chromosome). Two divergent lineages were created, namely 'Pure1' (n = 300) and 'Pure2' (n = 100), and gene flow between these two lineages was allowed to produce low levels of admixture. An additional set of 50 crossbred individuals was also included. The same dataset is available in the following formats: raw (equal to the phase format, except that the genotype matrix is not compressed), vcf and oxford. By using format = 'meta', metadata including pedigree and phenotypes for the 'Pure1' population can be downloaded. The pedigree contains 700 records, spanning 5 generations. The records in the phenotypes file are unbalanced repeated measurements (1 to 5 records per individual, with an average of 3) of a trait with heritability of 0.4, repeatability of 0.2, and a major QTL located on chromosome 3.

Since version 2.1.0, GHap maintains additional example files in its github repository (https://github.com/ytutsunomiya/GHap). In order to see which files are available, please see ghap.exfiles.

Author(s)

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

References

M. Sargolzaei and F. S. Schenkel. QMSim: A large-scale genome simulator for livestock. Bioinformatics. 2009. 25, 680–681.

Examples

```
# # See list of example files
# exlist <- ghap.exfiles()
# View(exlist)
#
# # Copy example data in phase format
# exfiles <- ghap.makefile()
# file.copy(from = exfiles, to = "./")
#
# # Copy example data in plink format
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink", verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy phenotypes and pedigree data
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta", verbose = TRUE)
# 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,
                  ncores = 1, verbose = TRUE)
```

Arguments

If all input files share the same prefix, the user can use the following shortcut options:

`input.files`	Character vector with the list of prefixes for input files.
`out.file`	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 or use multiple cores please use:

`ncores`	A numeric value specfying the number of cores to use (default = 1).
`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

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "oxford",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
#
# # Convert from a single genome-wide file
# ghap.oxford2phase(input.files = "example",
#                   out.file = "example")
#
# # Convert from a list of chromosome files
# ghap.oxford2phase(input.files = paste0("example_chr",1:10),
#                   out.file = "example")
#
# # Convert using separate lists for file extensions
# ghap.oxford2phase(haps.files = paste0("example_chr",1:10,".haps"),
#                   sample.files = paste0("example_chr",1:10,".sample"),
#                   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",1:10,".haps",sep="")
# command <- "tail -n+3 example_chr1.sample | cut -d' ' -f1,2 > example.samples"
# system(command)
# for(i in 1:10){
#   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.pedcheck *Summary statistics for pedigree*

Description

This function summarizes pedigree data and calculates inbreeding coefficients and equivalent complete generations.

Usage

```
ghap.pedcheck(ped, depth.n.f = FALSE)
```

Arguments

`ped` A dataframe with columns "id", "sire" and "dam" containing pedigree data.

`depth.n.f` A logical indicating if equivalent complete generations (depth) and inbreeding coefficients (f) should be calculated.

Value

A list containing two data frames: stats, which includes the pedigree summary; and ped, consisting of the original pedigree. If depth.n.f = TRUE, three columns are added to the ped data frame: gen (generation number), f (inbreding coefficient) and ecg (equivalent complete generations). The generation number and the inbreeding coefficient are computed with the help of the pedigreemm package.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

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 metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load pedigree data
# ped <- read.table(file = "example.pedigree", header=T)
#
# ### RUN ###
#
# # Descriptive statistics for the pedigree
# pedstat <- ghap.pedcheck(ped[,-1])
# print(pedstat$stats)
#
# # Retrieve inbreeding and pedigree depth
# pedstat <- ghap.pedcheck(ped[,-1], depth.n.f = TRUE)
# print(pedstat$ped)
# hist(pedstat$ped$f)
# hist(pedstat$ped$ecg)
```

ghap.phase2plink *Export phase object to PLINK binary*

Description

This function takes a phase object and converts it to the PLINK binary (bed/bim/fam) format.

Usage

```
ghap.phase2plink(object, out.file,
                 only.active.samples=TRUE,
                 only.active.markers=TRUE,
                 batchsize=NULL, ncores=1,
                 verbose=TRUE)
```

Arguments

`object` A GHap.phase object.

`out.file` A character value specifying the name used for the .bed, .bim and .fam output files.

`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 included in the output (default = TRUE).

`batchsize` A numeric value controlling the number of markers to be processed and written to output at a time (default = nmarkers/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

The returned output is a standard set of PLINK (Purcell et al., 2007; Chang et al., 2015) binary file (bed/bim/fam), meaning that phase information will be lost during conversion.

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.

Examples

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

```
# # Convert to plink
# ghap.phase2plink(object = phase, out.file = "example")
```

ghap.predictblup *Predict BLUP from reference*

Description

Prediction of BLUP values in test individuals based on reference individuals.

Usage

```
ghap.predictblup(refblup, vcp, covmat,
                 errormat = NULL,
                 errorname = "",
                 include.ref = TRUE,
                 diagonals = FALSE,
                 tol = 1e-12)
```

Arguments

refblup	A named numeric vector of reference BLUP values.
vcp	A numeric value for the variance in BLUP values.
covmat	A square matrix containing correlations among individuals. Both test and reference indiviudals must be present in the matrix.
errormat	A square error matrix for reference individuals. This matrix can be obtained with argument extras = "LHSi" in the ghap.lmm function.
errorname	The name used for the random effect in the ghap.lmm function. If the error matrix was imported from somewhere else, this argument can be ignored provided that the names in the error matrix match the ones in the covariance matrix.
include.ref	A logical value indicating if reference individuals should be included in the output (default = TRUE).
diagonals	A logical value indicating if diagonals of the covariance matrix should be used in calculations of accuracy and standard errors (default = FALSE). The default is to set diagonals to 1. For genomic estimated breeding values, using TRUE will account for inbreeding in the computation of accuracies and standard errors.
tol	A numeric value specifying the scalar to add to the diagonal of the covariance matrix if it is not inversible (default = 1e-12).

Value

A data frame with predictions of BLUP values. If an error matrix is provided, standard errors and accuracies are also included.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

J.F. Taylor. Implementation and accuracy of genomic selection. Aquaculture 2014. 420, S8-S14.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load phenotype and pedigree data
# df <- read.table(file = "example.phenotypes", header=T)
#
# ### RUN ###
#
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute genomic relationship matrix
# # Induce sparsity to help with matrix inversion
# K <- ghap.kinship(plink, sparsity = 0.01)
#
# # Fit mixed model
# df$rep <- df$id
# model <- ghap.lmm(formula = pheno ~ 1 + (1|id) + (1|rep),
#                   data = df,
#                   covmat = list(id = K, rep = NULL),
#                   extras = "LHSi")
# refblup <- model$random$id$Estimate
# names(refblup) <- rownames(model$random$id)
```

```
#
# # Predict blup of reference and test individuals
# blup <- ghap.predictblup(refblup, vcp = model$vcp$Estimate[1],
#                          covmat = as.matrix(K),
#                          errormat = model$extras$LHSi,
#                          errorname = "id")
#
# # Compare predictions
# plot(blup$Estimate, model$random$id$Estimate)
# abline(0,1)
```

ghap.profile *Genomic profile*

Description

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

Usage

```
ghap.profile(object, score, only.active.samples = TRUE,
             batchsize = NULL, ncores = 1, verbose = TRUE)
```

Arguments

`object` A valid GHap object (phase, haplo or plink).

`score` For HapAlleles (to use with GHap.haplo objects), the columns should be: BLOCK, CHR, BP1, BP2, ALLELE, SCORE, CENTER and SCALE. In the case of markers, the columns are: MARKER, CHR, BP, ALLELE, SCORE, CENTER and SCALE.

`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 or markers to be processed at a time (default = n/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 a reference 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 reference 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 HapAlleles or markers, the absent scores will be set to zero. If the input genomic data is a GHap.phase or GHap.plink object, the ALLELE column in the data frame is used as a guide to keep track of the direction of the scores. The default coding in GHap is A0/A0 = 0, A0|A1 = A1|A0 = 1 and A1/A1 = 2. For each marker where the declared allele is A0 instead of A1, that coding is flipped to A0/A0 = 2, A0|A1 = A1|A0 = 1 and A1/A1 = 1 so the profile is computed in the correct direction. Therefore, the user must be careful regarding the allele order, as well as the centering and scaling for each marker, since profiling is allele-oriented. 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 plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
```

```
#
# # Load phenotype and pedigree data
# df <- read.table(file = ”example.phenotypes”, header=T)
#
# ### RUN ###
#
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == ”Pure1”)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute genomic relationship matrix
# # Induce sparsity to help with matrix inversion
# K <- ghap.kinship(plink, sparsity = 0.01)
#
# # Fit mixed model
# df$rep <- df$id
# model <- ghap.lmm(formula = pheno ~ 1 + (1|id) + (1|rep),
#                   data = df,
#                   covmat = list(id = K, rep = NULL))
# refblup <- model$random$id$Estimate
# names(refblup) <- rownames(model$random$id)
#
# # Convert blup of individuals into blup of variants
# mkrblup <- ghap.varblup(object = plink,
#                         gebv = refblup,
#                         covmat = K)
#
# # Build GEBVs from variant effects and compare predictions
# gebv <- ghap.profile(object = plink, score = mkrblup)
# plot(gebv$SCORE, refblup); abline(0,1)
```

ghap.relfind *Find relatives in IBD estimates*

Description

This function infers relationships from IBD sharing.

Usage

```
ghap.relfind(ibdpairs, v = 50,
              breakclass = FALSE, ncores=1)
```

Arguments

`ibdpairs`	A dataframe containing IBD estimates, such as those provided by the output of –genome in plink.
`v`	A hyperparameter controling the variance of the likelihood functions (smaller values lead to more variance).
`breakclass`	A logical value indicating if relationship types 2 and 3 should be reported using their subclasses (default = FALSE).
`ncores`	A numeric value specifying the number of cores to be used in parallel computations (default = 1).

Details

The input dataframe must contain columns POP1 (or FID1), ID1 (or IID1), POP2 (or FID2), ID2 (or IID2), Z0, Z1, Z2 and PI_HAT. Columns Z0, Z1 and Z2 are the IBD sharing estimates representing the proportions of the genome where the two individuals being compared share exactly 0, 1 and 2 alleles identically by descent, respectively. The last column is Z2 + Z1/2, namely the proportion of the genome shared identically by descent.

This function implements a method based on composite likelihood scores to infer relationships based on the values of Z0, Z1, Z2 and PI_HAT. The predicted values are:

-1 = duplicates or monozygotic twins
0 = parent-offspring with inbreeding or self-fertilization
1 = parent-offspring
2 = full-siblings
3 = other types of relationships
4 = unrelated

Briefly, for each relationship type, the likelihood of each of the four IBD values is computed from a beta distribution with parameters a = m*v and b = (1-m)*v, where m is the expected value according to the relationship type and v is a hyperparameter controling the variance around the expected value (default v = 50). The composite likelihood is computed by summing the log-likelihoods for the four IBD values. The prediction is made by adopting the relationship type with the highest composite likelihood score. Details of the expected values of each relationship type is found in our vignette. This classification strategy is inspired by the method reported by Staples et al. (2014), albeit it is a different method. While Staples and collaborators infer relationships through Gaussian Kernel Density Estimation using only Z0 and Z1 values, our strategy uses the composite score formed by the sum of beta log-likelihoods for all IBD values.

An important detail is that relationship types 2 and 3 are in fact modelled through two and four different composite likelihoods, respectively, which are combined in order to achieve more robust and stable predictions. The user can choose to break down the subclasses via the argument 'breakclass = TRUE'. Reporting the subclasses is not default due to the reduced accuracy in distinguishing them. In addition, other relationship types not implicitly modelled in this version of the package may be confused with one of those subclasses. However, breaking these subclasses down may be useful for users seeking specific relationships in the data, as well as for those willing to perform pedigree simulations. The additional subclasses are:

2.1 = full-siblings
2.2 = full-siblings from a self-fertilized parent (or related parents)

3.1 = half-siblings, grandparent-grandchild or avuncular with inbreeding
3.2 = half-siblings, grandparent-grandchild or avuncular
3.3 = first-cousin or half-avuncular
3.4 = half-cousin and distant relatives

If the user chooses to run the analysis with 'breakclass = TRUE', beware that other types of cryptic relationships will be misclassified as pertaining to one of those four subclasses.

Value

The function returns the original dataframe with the extra column 'REL', containing the relationship predictions.

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.

J. Staples et al. PRIMUS: Rapid Reconstruction of Pedigrees from Genome-wide Estimates of Identity by Descent. 2014. 95, 553-564.

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 plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load pedigree data
# ped <- read.table(file = "example.pedigree", header=T)
#
# ### RUN ###
#
```

```
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == ”Pure1”)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute A1 allele frequencies
# p <- ghap.freq(plink, type = ”A1”)
#
# # Compute IBD statistics for individual 1
# pairlist <- data.frame(ID1 = pure1[1], ID2 = pure1[-1])
# ibd <- ghap.ibd(object = plink, pairlist = pairlist, freq = p,
#                  refsize = length(pure1))
#
# # Predict relationships for individual 1
# # 1 = parent-offspring
# # 3 = other types of relationship
# # 4 = unrelated
# rel <- ghap.relfind(ibdpairs = ibd)
# table(rel$REL)
#
# # Confirm with pedigree
# toprel <- rel$ID2[which(rel$REL == 1)]
# ped[which(ped$id %in% toprel & ped$dam == pure1[1]),]
```

ghap.remlci *Confidence intervals for functions of variance components*

Description

Approximation of standard errors and confidence intervals of arbitrary functions of variance components using a sampling-based method.

Usage

```
ghap.remlci(fun, vcp, ai, n = 10000,
             conf.level = 0.95,
             include.samples = FALSE,
             ncores = 1)
```

Arguments

`fun` Function of variance components. See details.

`vcp` A matrix or dataframe containing variance components, such as supplied by the `ghap.lmm` function.

`ai` The inverse of the average information matrix, such as supplied by the `ghap.lmm` function.

`conf.level` A numeric value informing the confidence level (default = 0.95).

`include.samples` A logical value indicating if samples of the likelihood function should be kept.

`n` A numerical value giving the number of samples to draw from the likelihood function (default = 10000).

`ncores` A numerical value specifying the number of cores to use in parallel computations (default = 1).

Details

The function implements the method of Meyer & Houle (2013) to approximate standard errors and confidence intervals of arbitrary functions of variance components. The method consists in assuming that REML estimates of variance components asymptotically follow a multivariate normal distribution with mean equals to the REML estimates themselves and covariance matrix equals the inverse of the average information matrix. Following that assumption, samples are drawn from that distribution and the user-defined function is applied to the samples. Standard errors are obtained as the standard deviation of the resulting vector, and confidence intervals are derived from vector quantiles.

The user must provide a function of variance components to the fun argument. The function has to be set assuming that the components are stored in a vector named x. For example, if variance components come from an animal model and the user wishes to obtain standard errors and confidence intervals for the heritability, simply use fun = function(x){x[1]/sum(x)}.

Value

The returned object is a list with the following items:

`stde` The standard error estimate.

`ci` The lower and upper limits of the confidence interval.

`samples` A vector containing the samples from the multivariate normal distribution. Only present in the output if the argument include.samples is set to TRUE.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

K. Meyer, D. Houle. Sampling based approximation of confidence intervals for functions of genetic covariance matrices. Proc. Assoc. Adv. Anim. Breed. 2013. 20, 523–527

J. Jensen et al. Residual maximum likelihood estimation of (Co)variance components in multivariate mixed linear models using average information. J. Ind. Soc. Ag. Statistics 1997. 49, 215-236.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load phenotype and pedigree data
# df <- read.table(file = "example.phenotypes", header=T)
#
# ### RUN ###
#
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute genomic relationship matrix
# # Induce sparsity to help with matrix inversion
# K <- ghap.kinship(plink, sparsity = 0.01)
#
# # Fit mixed model
# df$rep <- df$id
# model <- ghap.lmm(formula = pheno ~ 1 + (1|id) + (1|rep),
#                   data = df,
#                   covmat = list(id = K, rep = NULL))
#
# # Compute confidence interval for heritability
# h2 <- model$vcp$Estimate[1]/sum(model$vcp$Estimate)
# h2ci <- ghap.remlci(fun = function(x){x[1]/sum(x)},
#                     vcp = model$vcp, ai = model$AI)
# print(h2)
# print(h2ci)
```

ghap.roh *Detection of runs of homozygosity (ROH)*

Description

Map haplotype segments that are likely identical-by-descent.

Usage

```
ghap.roh(object, minroh = 1e+6, method = "hmm", freq = NULL,
         genpos = NULL, inbcoef = NULL, error = 0.25/100,
         only.active.samples = TRUE, only.active.markers = TRUE,
         ncores = 1, verbose = TRUE)
```

Arguments

The following arguments are used by both the 'hmm' and 'naive' methods:

`object` A valid GHap object (phase or plink).

`minroh` Minimum ROH length to output.

`method` Character value indicating which method to use: 'naive' or 'hmm' (default).

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

`only.active.markers` A logical value specifying whether only active markers should be used in the search (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).

The following arguments are only used by the 'hmm' method:

`freq` Named numeric vector of allele frequencies, such as provided by function `ghap.freq`.

`genpos` Named numeric vector of genetic positions. If not supplied, 1 cM = 1 Mb is assumed and genetic distances between consecutive markers are set to d/1e+6, where d is the distance in base pairs.

`inbcoef` Named numeric vector of starting values for genomic inbreeding (i.e., guess for the proportion of the genome covered by ROH).

`error` Numeric value representing the expected genotyping error rate (default = 0.25/100).

Details

This function searchs for runs of homozygosity (ROH) via two different methods:

The 'naive' method simply finds streches of homozygous genotypes in the observed haplotypes that are larger then a user-definied minimum size (default is 1 Mbp). The 'hmm' method uses a Hidden Markov Model that takes genotyping error and recombination into account while detecting ROHs. The 'hmm' model in GHap is similar to the ones described by Narasimhan et al (2016) and Druet & Gautier (2017), differing slightly in model fitting and definition of transition and emission probabilities (details are covered in our vignette).

The 'hmm' method requires allele frequencies for each marker, as well as starting values for the expected proportion of the genome covered by ROH (genomic inbreeding) for each individual. Estimates of allele frequencies can be either based on a reference or estimated from the data with the `ghap.freq` function. Starting values for genomic inbreeding can be obtained by running the function with the 'naive' method first and then computing starting values with `ghap.froh` (see the examples). A genetic map with positions in cM can be provided by the user via the genpos argument. If genetic positions are not provided, 1 cM = 1 Mb is assumed.

Value

The function returns a dataframe with the following columns:

`POP`	Original population label.
`ID`	Individual name.
`CHR`	Chromosome name.
`BP1`	Segment start position.
`BP2`	Segment end position.
`LENGTH`	Length of run of homozygosity.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

References

V. Narasimhan et al. BCFtools/RoH: a hidden Markov model approach for detecting autozygosity from next-generation sequencing data. Bioinformatics. 2016. 32:1749-1751.

T. Druet & M. Gautier. A model-based approach to characterize individual inbreeding at both global and local genomic scales. Molecular Ecology. 2017. 26:5820-5841.

See Also

`ghap.freq`, `ghap.froh`

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# ### RUN ###
#
# # Subset pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # ROH via the 'naive' method
# roh1 <- ghap.roh(plink, method = "naive")
# froh1 <- ghap.froh(plink, roh1)
#
# # ROH via the 'hmm' method
# freq <- ghap.freq(plink, type = 'A1')
# inbcoef <- froh1$FROH1; names(inbcoef) <- froh1$ID
# roh2 <- ghap.roh(plink, method = "hmm", freq = freq,
#                  inbcoef = inbcoef)
# froh2 <- ghap.froh(plink, roh2)
#
# # Method 'hmm' using Fhat3 as starting values
# inbcoef <- ibc$Fhat3; names(inbcoef) <- ibc$ID
# inbcoef[which(inbcoef < 0)] <- 0.01
# roh3 <- ghap.roh(plink, method = "hmm", freq = freq,
#                   inbcoef = inbcoef)
# froh3 <- ghap.froh(plink, roh3)
```

---

ghap.simadmix *Simulate individuals from specified admixture proportions*

Description

Generation of simulated haplotypes based on a list of user-defined ancestral populations.

Usage

```
ghap.simadmix(object, n.individuals,
              n.generations, ancestors,
              proportions = NULL,
              alpha = NULL,
              out.file,
              only.active.markers = TRUE,
              ncores = 1,verbose = TRUE)
```

Arguments

`object` A GHap.phase object.

`n.individuals` Number of individuals to simulate.

`n.generations` Number of generations past the admixture event.

`ancestors` List of ancestral populations. Each ancestral population is given as a vector of ids of the ancestors belonging to that population.

`proportions` A dataframe containing the ancestry proportions in the simulated individuals. The number of columns has to be equal to the number of ancestral populations defined in the ancestors list, and the number of row has to be equal to the number of simulated individuals. See argument 'alpha' if you want the function to generate random samples for admixture proportions.

`alpha` A list with same size of the 'ancestors' list, with each value representing the vector of parameters to be used for sampling ancestry proportions from a Direchelet distribution.

`out.file` Output file name.

`only.active.markers` A logical value specifying whether only active markers should be used in simulations (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

Given a list of ancestral populations, this function simulates haplotypes of individuals descending from admixture among these populations after a defined number of generations. The ancestry proportions can be sampled from a Direchelet distribution with a vector of parameters alpha. For example, if three ancestral populations are considered and the parameters are set as alpha = list(pop1 = 1, pop2 = 1, pop3 = 1), the resulting simulated individuals will have a highly diverse configuration of ancestry proportions. On the other hand, by setting alpha = list(pop1 = 0, pop2 = 1, pop3 = 0) for example, all individuals will descend entirely from population 2 without admixture. The user can play with these values to fine tune the desired sampling distribution. Alternatively, exact proportions for each simulated individual can be set through the 'proportions' argument by providing a dataframe containing the desired ancestry values.

Value

The function outputs the following files:

- **.samples**: space-delimited file without header containing two columns: Population and ID of simulated individuals (numbered from 1 to n). The first column is filled with "SIM".
- **.markers**: space-delimited file without header containing five columns: Chromosome, Marker, Position (in bp), Reference Allele (A0) and Alternative Allele (A1).
- **.phase**: space-delimited file without header containing the phased genotype matrix of the simulated progeny. The dimension of the matrix is *m x 2n*, where *m* is the number of markers and *n* is the number of simulated individuals (i.e., two columns per individual, representing the two phased chromosome alleles).
- **.proportions**: space-delimited file with header containing the following columns: Population, ID and K columns of ancestry proportions, one for each ancestral population.
- **.haplotypes**: space-delimited file with header containing ancestry tracks with the following columns: Population, ID, haplotype number, chromosome name, starting position, ending position, track size and ancestry of the segment.

The user can then treat these files as a regular GHap input, or use them to build the Oxford HAPS/SAMPLES format for analysis with other software. The simulated data can be useful in the evaluation of accuracy of different algorithms designed for admixture analysis, as well as of other methods in the field of population genomics.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
# phase <- ghap.loadphase("example")
#
# # Make vectors of ancestors
# pure1 <- unique(phase$id[which(phase$pop == "Pure1")])
# pure2 <- unique(phase$id[which(phase$pop == "Pure2")])
#
# # Simulate proportions
# ngroup <- 30
# pop1 <- c(rep(1, times = ngroup), # purebred from population 1
#           runif(n = ngroup, min = 0.1, max = 0.9), # admixed individuals
#           rep(0, times = ngroup)) # purebred from population 2
# pop2 <- 1-pop1
# prop <- data.frame(pop1,pop2)
#
# # Simulate individuals
# set.seed(1988)
# ghap.simadmix(object = phase, n.individuals = nrow(prop),
#               n.generations = 10,
#               ancestors = list(pop1 = pure1, pop2 = pure2),
#               proportions = prop, out.file = "sim")
# ghap.compress(input.file = "sim", out.file = "sim")
# sim <- ghap.loadphase("sim")
#
# # Unsupervised analysis with K = 2
# prototypes <- ghap.anctrain(object = sim, K = 2)
# hapadmix <- ghap.anctest(object = sim,
#                          prototypes = prototypes,
#                          test = unique(sim$id))
# anctracks <- ghap.ancsmooth(object = sim, admix = hapadmix)
#
# # Load simulated ancestry proportions
# ancsim <- NULL
# ancsim$proportions2 <- read.table(file = "sim.proportions", header=T)
# ancsim$haplotypes <- read.table(file = "sim.haplotypes", header=T)
#
# # Compare estimates with real values
# # Obs: the original populations had introgression from each other
# # Admixture seen in the estimates of purebreds reflect that introgression
# ghap.ancplot(ancsmooth = ancsim)
# ghap.ancplot(ancsmooth = anctracks)
# cor(anctracks$proportions2$K1, ancsim$proportions2$pop1)
# ghap.karyoplot(ancsmooth = ancsim, ids = sim$id[66])
# ghap.karyoplot(ancsmooth = anctracks, ids = sim$id[66])
```

ghap.simmating *Simulate individuals from specified matings*

Description

Generation of simulated haplotypes based on in silico matings.

Usage

```
ghap.simmating(object, n.individuals = 1,
               parent1 = NULL, parent2 = NULL,
               model = "proportional", out.file,
               only.active.markers = TRUE,
               ncores = 1, verbose = TRUE)
```

Arguments

object	A GHap.phase object.
n.individuals	Number of individuals to simulate.
parent1	Vector containing the ids of candidate sires. When a character vector is provided, all candidates are considered equally likely to be selected. Alternatively, a named numeric vector of probabilities (with sire ids used as names) can be used in order to make specific sires more likely to be selected.

`parent2` Vector containing the ids of candidate dams. When a character vector is provided, all candidates are considered equally likely to be selected. Alternatively, a named numeric vector of probabilities (with dam ids used as names) can be used in order to make specific dams more likely to be selected.

`model` The model used for sampling the number of recombinations per chromosome (default = "proportional"). See details for more options.

`out.file` Output file name.

`only.active.markers`
A logical value specifying whether only active markers should be used in simulations (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

Given a list of candidate parents, this function samples sire i with probability p_i and dam j with probability p_j (these probabilities should be provided by the user, otherwise matings will occur at random). Once sire and dam are sampled, gametes are simulated by creating recombinant parental haplotypes. The progeny is then obtained by uniting the simulated gametes.

The default model ("proportional") assumes that the number of recombinations per meiosis across all chromosomes follows a Poisson distribution with mean equal to nchr (the number of chromosome pairs), such that the number of recombinations for a given chromosome is sampled from a Poisson distribution with mean prop*nchr, where prop is the proportion of the genome size covered by that chromosome. Therefore, this option takes chromosome size in consideration while sampling the number of recombination events. In option "uniform", the number of recombinations is sampled from a Poisson distribution with mean 1 for each chromosome instead. Alternatively, the model argument can also take a named vector of chromosome-specific recombination rates in cM/Mb. In this option, the number of recombinations for a given chromosome is sampled from a Poisson distribution with mean chrsize*chrrate/100, where chrsize is the size of the chromosome in Mb and chrrate is the recombination rate in cM/Mb. A last option is offered where the model argument is a named vector of marker-specific recombination rates in cM/Mb. The number of recombinations for a given chromosome is also sampled from a Poisson distribution with mean chrsize*chrrate/100, but with chrrate calculated as the average across markers within the same chromosome. In this model, instead of placing the recombination breakpoint randomly within a chromosome for each gamete, marker-specific recombination rates are taken into account, making regions in the chromosome with higher recombination rates more susceptible to breaks.

Value

The function outputs the following files:

- **.samples**: space-delimited file without header containing two columns: Population and ID of simulated individuals (numbered from 1 to n). The first column is filled with "SIM".
- **.markers**: space-delimited file without header containing five columns: Chromosome, Marker, Position (in bp), Reference Allele (A0) and Alternative Allele (A1).

- **.phase**: space-delimited file without header containing the phased genotype matrix of the simulated progeny. The dimension of the matrix is *m x 2n*, where *m* is the number of markers and *n* is the number of simulated individuals (i.e., two columns per individual, representing the two phased chromosome alleles).
- **.pedrigree**: space-delimited file without header containing three columns: individual number, sire ID and dam ID.

The user can then treat these files as a regular GHap input, or use them to build the Oxford HAPS/SAMPLES format for analysis with other software. The simulated data can be useful in the evaluation of mating plans, since the in silico progeny can help in the characterization of the expected distribution of genomic inbreeding coefficients, ancestry proportions and EBVs in the progeny. Since the function does not check for sex differences between parents, the user can perform simulations of self-fertilization (relevant for plant breeders) and same sex matings.

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(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Simulation using only two specific parents
# parent1 <- phase$id[1]
# parent2 <- phase$id[3]
# ghap.simmating(phase, n.individuals = 100,
#                parent1 = parent1, parent2 = parent2,
#                out.file = "sim1", ncores = 1)
#
# # Simulation using candidates with unequal probabilities
# parent1 <- c(0.5,0.25,0.25)
# names(parent1) <- phase$id[c(1,3,5)]
# parent2 <- c(0.7,0.2,0.1)
# names(parent2) <- phase$id[c(7,9,11)]
# ghap.simmating(phase, n.individuals = 100,
#                parent1 = parent1, parent2 = parent2,
#                out.file = "sim2", ncores = 1)
```

ghap.simpheno *Quantitative trait simulation using real genotype data*

Description

Simulates phenotypes from a quantitative trait with arbitrary variant-specific heritabilities.

Usage

```
ghap.simpheno(object, h2, r2 = 0, nrep = 1,
              balanced = TRUE, seed = NULL,
              only.active.samples = TRUE,
              ncores = 1, verbose = TRUE)
```

Arguments

`object`	A valid GHap object (phase or plink).
`h2`	A named numeric value specifying the heritability per variant. The sum of variant-specific heritabilities will be set as the narrow-sense heritability, and must not exceed 1.
`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.
`seed`	A numeric value used to set the random number generation state (default = NULL). This is useful for reproducibility of the results.
`only.active.samples`	A logical value specifying whether only active samples should be used for calculations (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 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. True breeding values are computed from the sum of causal variant effects specified in the 'h2' argument. Both the residual and permanent environmental effects are sampled from normal distributions.

Value

The function returns a data frame with items:

POP	Original population label.
ID	Individual name.
PHENO	Phenotypic observation.
TBV	True breeding value.
REP	Permanent environmental effect (only present if nrep > 1).
RESIDUAL	Residual value.

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(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load phase data
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Subset Pure1 population
# pure1 <- unique(phase$id[which(phase$pop == "Pure1")])
# phase <- ghap.subset(object = phase, ids = pure1,
#                      variants = phase$marker)
# freq <- ghap.freq(object = phase, type = "maf")
#
# # Heritability   = 0.3
# # Number of QTLs = 1000
# # Major QTLs     = 0
# # Records per id = 1
# nqtl <- 1000
# h2 <- 0.3
# mkr <- sample(names(freq[which(freq > 0.05)]), size = nqtl)
# eff <- runif(n = nqtl, min = 0, max = 1)
# eff <- h2*eff/sum(eff)
# names(eff) <- mkr
# df1 <- ghap.simpheno(object = phase, h2 = eff)
#
# # Heritability   = 0.5
# # Number of QTLs = 100
# # Major QTLs     = 1
# # Records per id = 5 (balanced)
# # Repeatability = 0.2
# nqtl <- 100
# h2 <- 0.4
# r2 <- 0.2
# reps <- 5
# mkr <- sample(names(freq[which(freq > 0.05)]), size = nqtl)
# eff <- runif(n = nqtl, min = 0, max = 1)
# eff <- h2*eff/sum(eff)
# eff[which(eff == min(eff))] <- 0.1
# names(eff) <- mkr
# df2 <- ghap.simpheno(object = phase, h2 = eff,
#                      r2 = r2, nrep = reps)
#
# # Heritability   = 0.5
# # Number of QTLs = 100
# # Major QTLs     = 1
# # Records per id = 5 (unbalanced)
# # Repeatability = 0.2
# nqtl <- 100
# h2 <- 0.4
# r2 <- 0.2
# reps <- 5
# mkr <- sample(names(freq[which(freq > 0.05)]), size = nqtl)
# eff <- runif(n = nqtl, min = 0, max = 1)
# eff <- h2*eff/sum(eff)
# eff[which(eff == min(eff))] <- 0.1
# names(eff) <- mkr
# df3 <- ghap.simpheno(object = phase, h2 = eff, r2 = r2,
#                      nrep = reps, balanced = FALSE)
```

ghap.slice *Get a slice of a GHap object*

Description

This function parses a binary PLINK, phased or HapGenotypes matrix and returns the slice as an R matrix.

Usage

```
ghap.slice(object, ids, variants, index=FALSE,
           transposed=FALSE, sparse=TRUE,
           unphase=FALSE, impute=FALSE,
           counted="A1")
```

Arguments

`object` A valid GHap object (phase, haplo or plink).

`ids` A character or numeric vector indicating individuals to parse.

`variants` A character or numeric vector indicating variants to parse. If a "GHap.haplo" object is provided, the vector must be numeric.

`index` A logical value specfying if values provided for ids and variants are indices (see details).

`transposed` A logical value specfying if genotypes should be transposed. If FALSE (default), the matrix is returned as variants by individuals. Otherwise, the retrieved matrix is organized as individuals by variants.

`sparse` A logical value specfying if the returned matrix should be formatted as a sparse matrix (default TRUE).

`unphase` A logical value specfying if phased genotypes should be retrieved as unphased allele counts. Only meaningful for "GHap.phase" objects.

`impute` A logical value specfying if missing genotypes should be replaced by 0 (i.e., A0/A0 genotypes, default = FALSE). Only meaningful for "GHap.plink" objects.

`impute` A character value specfying the allele to be counted (i.e., A0 or A1, default = "A1").

Details

This function parses the binary input file and returns an R matrix with the requested list of variants (markers or 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 object (index = TRUE). In the case of "GHap.haplo" objects, HapAlleles can only by parsed via indices.

Value

An R matrix with variants in rows and individuals in columns (this is inverted if transposed = TRUE).

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>, Adam Taiti Harth Utsunomiya <adamtaiti@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load data
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Select random individuals and markers
# ind <- sample(x = unique(phase$id), size = 5)
# mkr <- sample(x = phase$marker, size = 10)
#
# # Generate slice of the data
# ghap.slice(object = phase, ids = ind, variants = mkr)
#
# # Import as unphased data
# ghap.slice(object = phase, ids = ind, variants = mkr,
#            unphase = TRUE)
#
# # Return transposed matrix
# ghap.slice(object = phase, ids = ind, variants = mkr,
#            unphase = TRUE, transposed = TRUE)
#
# # Display data as non-sparse matrix
# ghap.slice(object = phase, ids = ind, variants = mkr,
#            unphase = TRUE, transposed = TRUE,
#            sparse = FALSE)
```

ghap.subset *Subset GHap objects*

Description

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

Usage

```
ghap.subset(object, ids, variants,
            index=FALSE, verbose=TRUE)
```

Arguments

object	A valid GHap object (phase, haplo or plink).
ids	A character or numeric vector indicating individuals to parse.
variants	A character or numeric vector indicating variants to parse. If a "GHap.haplo" object is provided, the vector must be numeric.
index	A logical value specfying if values provided for ids and variants are indices (see details).
verbose	A logical value specfying whether log messages should be printed (default = TRUE).

Details

This function sets to FALSE (i.e., inactivates) all individuals and variants not included in the provided arguments. This procedure avoids expensive subsetting operations by simply flagging which variants and individuals should be used in downstream analyses. The argument index allows the user to specify individuals either by name (index = FALSE) or by indices as stored in the GHap object (index = TRUE). In the case of "GHap.haplo" objects, HapAlleles can only by parsed via indices.

Value

A GHap object of the same type as the one used in the object argument.

Author(s)

Yuri Tani Utsunomiya <ytutsunomiya@gmail.com>

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy phase data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "phase",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load data
# phase <- ghap.loadphase("example")
#
# ### RUN ###
#
# # Subset individuals from population 'Pure1'
# pure1 <- unique(phase$id[which(phase$pop == "Pure1")])
# phase <- ghap.subset(object = phase, ids = pure1,
#                      variants = phase$marker)
#
# # Calculate allele frequencies for population 'Pure1'
# freq <- ghap.freq(phase, type = 'maf')
#
# # Subset markers with MAF > 0.05 in population 'Pure1'
# mkr <- names(freq)[which(freq > 0.05)]
# phase <- ghap.subset(object = phase, ids = pure1,
#                      variants = mkr)
```

ghap.varblup *Convert BLUP of individuals into BLUP of variants*

Description

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

Usage

```
ghap.varblup(object, gebv, covmat,
             type = 1, batchsize = NULL,
             only.active.variants = TRUE,
             weights = NULL, tol = 1e-12,
             vcp = NULL, errormat = NULL,
             errorname = "", invcov = FALSE,
             nlambda = 1000, ncores = 1,
             verbose = TRUE)
```

Arguments

`object` A valid GHap object (phase, haplo or plink).

`gebv` A named vector of genomic estimated breeding values.

`covmat` An additive genomic relationship matrix, such as obtained with type=1 or type=2 in the ghap.kinship function.

`type` A numeric value indicating the type of relationship matrix (see details in the ghap.kinship function).

`batchsize` A numeric value controlling the number of variants to be processed at a time (default = nalleles/10).

`only.active.variants`
A logical value specifying whether only active variants should be included in the calculations (default = TRUE).

`weights` A numeric vector providing variant-specific weights.

`tol` A numeric value specifying the scalar to add to the diagonal of the relationship matrix it is not inversible (default = 1e-10).

`vcp` A numeric value for the variance in GEBVs.

`errormat` A square error matrix for GEBVs. This matrix can be obtained with argument extras = "LHSi" in the ghap.lmm function. If provided, calculation of standard errors and test statistics for the variants is activated.

`errorname` The name used for the random effect representing GEBVs in the ghap.lmm function. If the error matrix was imported from somewhere else, this argument can be ignored provided that the names in the error matrix match the ones in the relationship matrix.

`invcov` A logic value indicating whether the provided covariance matrix is an inverse.

`nlambda` A numeric value for the number of variants to be used in the estimation of the inflation factor (default = 1000).

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

`verbose` A logical value specifying 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 *m* matrix of genotypes, where *n* is the number of individuals and *m* is the number of variants (i.e, markers or HapAlleles), $\mathbf{D} = diag(d_i)$ with d_i being the weight of variant *i* (default $d_i = 1$), q is the inverse weighted sum of variances in the columns of $\mathbf{M}$, $\mathbf{K}$ is the additive genomic relationship matrix and $\hat{u}$ is the vector of GEBVs.

Value

The function returns a data frame with results from the genome-wide conversion of BLUP of individuals into BLUP of variants. If a GHap.haplo object is used, the first columns of the data frame will be:

`CHR`	Chromosome name.
`BLOCK`	Block alias.
`BP1`	Block start position.
`BP2`	Block end position.

For GHap.phase and GHap.plink objects, the first columns will be:

`CHR`	Chromosome name.
`MARKER`	Block start position.
`BP`	Block end position.

The remaining columns of the data frame will be equal for any class of GHap objects:

`ALLELE`	Identity of the counted (A1 or haplotype) allele.
`FREQ`	Frequency of the allele.
`SCORE`	Estimated BLUP of the allele.
`VAR`	Variance in allele-specific breeding values.
`pVAR`	Proportion of variance explained by the allele.
`CENTER`	Average genotype (meaningful only for predictions with `ghap.profile`).
`SCALE`	A constant set to 1 (meaningful only for predictions with `ghap.profile`).

If an error matrix for GEBVs is provided through the 'errormat' argument, the following additional columns are included in the data frame:

`SE` Standard error for the BLUP of the allele.

`CHISQ.EXP` Expected values for the test statistics.

`CHISQ.OBS` Observed value for the test statistics.

`CHISQ.GC` Test statistics scaled by the inflation factor (Genomic Control). Inflation is computed through regression of observed quantiles onto expected quantiles. In order to avoid overestimation by variants rejecting the null hypothesis, a random sample of variants (with size controled via the nlambda argument) is taken within three standard deviations from the mean of the distribution of test statistics.

`LOGP` log10(1/P) or -log10(P) for the BLUP of the allele.

`LOGP.GC` log10(1/P) or -log10(P) for the BLUP of the allele (scaled by the inflation factor).

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.

J.L.G. Duarte et al. Rapid screening for phenotype-genotype associations by linear transformations of genomic evaluations. BMC Bioinformatics. 2014, 15:246.

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy plink data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "plink",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Copy metadata in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "meta",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# # Load plink data
# plink <- ghap.loadplink("example")
#
# # Load phenotype and pedigree data
# df <- read.table(file = "example.phenotypes", header=T)
#
# ### RUN ###
#
# # Subset individuals from the pure1 population
# pure1 <- plink$id[which(plink$pop == "Pure1")]
# plink <- ghap.subset(object = plink, ids = pure1, variants = plink$marker)
#
# # Subset markers with MAF > 0.05
# freq <- ghap.freq(plink)
# mkr <- names(freq)[which(freq > 0.05)]
# plink <- ghap.subset(object = plink, ids = pure1, variants = mkr)
#
# # Compute genomic relationship matrix
# # Induce sparsity to help with matrix inversion
# K <- ghap.kinship(plink, sparsity = 0.01)
#
# # Fit mixed model
# df$rep <- df$id
# model <- ghap.lmm(formula = pheno ~ 1 + (1|id) + (1|rep),
#                   data = df,
#                   covmat = list(id = K, rep = NULL),
#                   extras = "LHSi")
# refblup <- model$random$id$Estimate
# names(refblup) <- rownames(model$random$id)
#
# # Convert blup of individuals into blup of variants
# mkrblup <- ghap.varblup(object = plink, gebv = refblup,
#                         covmat = K, vcp = model$vcp$Estimate[1],
#                         errormat = model$extras$LHSi, errorname = "id")
#
# # Build GEBVs from variant effects and compare predictions
# gebv <- ghap.profile(object = plink, score = mkrblup)
# plot(gebv$SCORE, refblup); abline(0,1)
#
# # Compare variant solutions with regular GWAS
# gwas <- ghap.assoc(object = plink,
#                    formula = pheno ~ 1 + (1|id) + (1|rep),
#                    data = df,
#                    covmat = list(id = K, rep = NULL))
# ghap.manhattan(data = gwas, chr = "CHR", bp = "BP", y = "LOGP")
# ghap.manhattan(data = mkrblup, chr = "CHR", bp = "BP", y = "LOGP")
# plot(mkrblup$LOGP, gwas$LOGP); abline(0,1)
```

---

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

Arguments

If all input files share the same prefix, the user can use the following shortcut options:

input.files	Character vector with the list of prefixes for input files.
out.file	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 SAMPLE files.

To turn conversion progress-tracking on or off or set the number of cores please use:

ncores	A numeric value specfying the number of cores to use (default = 1).
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 very 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

Examples

```
# #### DO NOT RUN IF NOT NECESSARY ###
#
# # Copy the example data in the current working directory
# exfiles <- ghap.makefile(dataset = "example",
#                          format = "vcf",
#                          verbose = TRUE)
# file.copy(from = exfiles, to = "./")
#
# ### RUN ###
#
# # Convert from a single genome-wide file
# ghap.vcf2phase(input.files = "example",
#                out.file = "example")
#
# # Convert from a list of chromosome files
# ghap.vcf2phase(input.files = paste0("example_chr",1:10),
#                out.file = "example")
#
# # Convert using separate lists for file extensions
# ghap.vcf2phase(vcf.files = paste0("example_chr",1:10,".vcf"),
#                sample.files = paste0("example_chr",1:10,".sample"),
#                out.file = "example")
```

Index