Package ‘SNPRelate’

January 31, 2017

Type Package

Title Parallel Computing Toolset for Relatedness and Principal Component Analysis of SNP Data

Version 1.8.0

Date 2016-10-13

Depends R (>= 2.15), gdsfmt (>= 1.8.3)

LinkingTo gdsfmt

Suggests parallel, RUnit, knitr, MASS, BiocGenerics

Enhances SeqArray (>= 1.11.12)

Description Genome-wide association studies (GWAS) are widely used to investigate the genetic basis of diseases and traits, but they pose many computational challenges. We developed an R package SNPRelate to provide a binary format for single-nucleotide polymorphism (SNP) data in GWAS utilizing CoreArray Genomic Data Structure (GDS) data files. The GDS format offers the efficient operations specifically designed for integers with two bits, since a SNP could occupy only two bits. SNPRelate is also designed to accelerate two key computations on SNP data using parallel computing for multi-core symmetric multiprocessing computer architectures: Principal Component Analysis (PCA) and relatedness analysis using Identity-By-Descent measures. The SNP GDS format is also used by the GWASTools package with the support of S4 classes and generic functions. The extended GDS format is implemented in the SeqArray package to support the storage of single nucleotide variations (SNVs), insertion/deletion polymorphism (indel) and structural variation calls.

License GPL-3

VignetteBuilder knitr

URL http://github.com/zhengxwen/SNPRelate,
http://corearray.sourceforge.net/tutorials/SNPRelate/

BugReports http://github.com/zhengxwen/SNPRelate/issues

biocViews Infrastructure, Genetics, StatisticalMethod, PrincipalComponent

NeedsCompilation yes
Author Xiuwen Zheng [aut, cre, cph],
Stephanie Gogarten [ctb],
Cathy Laurie [ctb],
Bruce Weir [ctb, ths]

Maintainer Xiuwen Zheng <zhengx@u.washington.edu>

R topics documented:

SNPRelate-package .................................................. 3
hapmap_geno ....................................................... 5
snpgdsAdmixProp .................................................... 6
snpgdsAlleleSwitch .................................................. 8
snpgdsApartSelection ............................................... 9
snpgdsBED2GDS ...................................................... 10
snpgdsClose ......................................................... 12
snpgdsCombineGeno .................................................. 13
snpgdsCreateGeno ................................................... 14
snpgdsCreateGenoSet ............................................... 15
snpgdsCutTree ....................................................... 17
snpgdsDiss ........................................................ 19
snpgdsDrawTree ..................................................... 21
snpgdsEIGMIX ........................................................ 22
snpgdsErrMsg ......................................................... 24
snpgdsExampleFileName .............................................. 25
SNPGDSFileClass ................................................... 26
snpgdsFst .......................................................... 26
snpgdsGDS2BED ....................................................... 28
snpgdsGDS2Eigen ...................................................... 29
snpgdsGDS2PED ....................................................... 30
snpgdsGEN2GDS ....................................................... 31
snpgdsGetGeno ....................................................... 33
snpgdsGRM .......................................................... 34
snpgdsHCluster ...................................................... 35
snpgdsHWE ........................................................... 37
snpgdsIBDKING ....................................................... 38
snpgdsIBDMLE ........................................................ 40
snpgdsIBDMLELogLik .................................................. 43
snpgdsIBDMoM ......................................................... 45
snpgdsIBDSelection .................................................. 47
snpgdsIBS ............................................................ 49
snpgdsIBSNnum ....................................................... 50
snpgdsIndInb ......................................................... 51
snpgdsIndInbCoef ..................................................... 53
snpgdsIndivBeta ...................................................... 54
snpgdsLDMat ........................................................ 55
snpgdsLdpair ........................................................ 57
snpgdsLdpruning ...................................................... 58
snpgdsOpen .......................................................... 60
snpgdsOption ........................................................ 61
snpgdsPairIBD ......................................................... 62
snpgdsPairIBDMLELogLik .............................................. 64
Genome-wide association studies are widely used to investigate the genetic basis of diseases and traits, but they pose many computational challenges. We developed SNPRelate (R package for multi-core symmetric multiprocessing computer architectures) to accelerate two key computations on SNP data: principal component analysis (PCA) and relatedness analysis using identity-by-descent measures. The kernels of our algorithms are written in C/C++ and highly optimized.

The genotypes stored in GDS format can be analyzed by the R functions in SNPRelate, which utilize the multi-core feature of machine for a single computer.


Tutorial: http://corearray.sourceforge.net/tutorials/SNPRelate/

Author(s)

Xiuwen Zheng <zhengxwen@gmail.com>
References


Examples

####################################################################
# Convert the PLINK BED file to the GDS file
#
# # PLINK BED files
bed.fn <- system.file("extdata", "plinkhapmap.bed.gz", package="SNPRelate")
fam.fn <- system.file("extdata", "plinkhapmap.fam.gz", package="SNPRelate")
bim.fn <- system.file("extdata", "plinkhapmap.bim.gz", package="SNPRelate")

# convert
snpgdsBED2GDS(bed.fn, fam.fn, bim.fn, "HapMap.gds")

####################################################################
# Principal Component Analysis
#
# # open
genofile <- snpgdsOpen("HapMap.gds")
RV <- snpgdsPCA(genofile)
plot(RV$eigenvect[,2], RV$eigenvect[,1], xlab="PC 2", ylab="PC 1",
     col=rgb(0,0,150, 50, maxColorValue=255), pch=19)

# close the file
snpgdsClose(genofile)

####################################################################
# Identity-By-Descent (IBD) Analysis
#
# # open
genofile <- snpgdsOpen(snporgdsExampleFileName())
RV <- snpgdsIBDMoM(genofile)
flag <- lower.tri(RV$k0)
plot(RV$k0[flag], RV$k1[flag], xlab="k0", ylab="k1",
     col=rgb(0,0,150, 50, maxColorValue=255), pch=19)
abline(1, -1, col="red", lty=4)

# close the file
snpgdsClose(genofile)

####################################################################
# Identity-By-State (IBS) Analysis
#

SNPRelate-package
# open
genofile <- snpgdsOpen(snpgdsExampleFileName())

RV <- snpgdsIBS(genofile)
m <- 1 - RV$ibs
colnames(m) <- rownames(m) <- RV$sample.id
GeneticDistance <- as.dist(m[1:45, 1:45])
HC <- hclust(GeneticDistance, "ave")
plot(HC)

# close the file
snpgdsClose(genofile)

# Linkage Disequilibrium (LD) Analysis
#
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

snpset <- read.gdsn(index.gdsn(genofile, "snp.id"))[1:200]
L1 <- snpgdsLDMat(genofile, snp.id=snpset, method="composite", slide=-1)

# plot
image(abs(L1$LD), col=terrain.colors(64))

# close the file
snpgdsClose(genofile)

__hapmap_geno__  
**SNP genotypes of HapMap samples**

**Description**

A list object including the following components:
sample.id – a vector of sample ids;
snp.id – a vector of SNP ids;
snp.position – a vector of SNP positions;
snp.chromosome – a vector of chromosome indices;
snp.allele – a character vector of “reference / non-reference”;
genotype – a “# of SNPs” X “# of samples” genotype matrix.

**Usage**

hapmap_geno

**Value**

A list
snpgdsAdmixProp

Estimate ancestral proportions from the eigen-analysis

Description

Estimate ancestral (admixture) proportions based on the eigen-analysis.

Usage

snpgdsAdmixProp(eigobj, groups, bound=FALSE)

Arguments

eigobj an object of snpgdsEigMixClass from snpgdsEIGMIX, or an object of snpgdsPCAClass from snpgdsPCA

groups a list of sample IDs, such like groups = list(CEU = c("NA0101", "NA1022", ...), YRI = c("NAxxxx", ...), Asia = c("NA1234", ...))

bound if TRUE, the estimates are bounded so that no component < 0 or > 1, and the sum of proportions is one

Details

The minor allele frequency and missing rate for each SNP passed in snp.id are calculated over all the samples in sample.id.

Value

Return a snpgdsEigMixClass object, and it is a list:

sample.id the sample ids used in the analysis
snp.id the SNP ids used in the analysis
eigenval eigenvalues
eigenvect eigenvectors, "# of samples" x "eigen.cnt"
ibdmat the IBD matrix

Author(s)

Xiuwen Zheng

References


See Also

snpgdsEIGMIX, snpgdsPCA
Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpDgsExampleFileName())

# get population information
# or pop_code <- scan("pop.txt", what=character())
# if it is stored in a text file "pop.txt"
pop_code <- read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))

# get sample id
samp.id <- read.gdsn(index.gdsn(genofile, "sample.id"))

# run eigen-analysis
RV <- snpgdsEIGMIX(genofile)

# eigenvalues
RV$eigenval

# make a data.frame
tab <- data.frame(sample.id = samp.id, pop = factor(pop_code),
                  EV1 = RV$eigenvect[,1], # the first eigenvector
                  EV2 = RV$eigenvect[,2], # the second eigenvector
                  stringsAsFactors = FALSE)
head(tab)

# draw
plot(tab$EV2, tab$EV1, col=as.integer(tab$pop),
     xlab="eigenvector 2", ylab="eigenvector 1")
legend("topleft", legend=levels(tab$pop), pch="o", col=1:4)

# define groups
groups <- list(CEU = samp.id[pop_code == "CEU"],
               YRI = samp.id[pop_code == "YRI"],
               CHB = samp.id[is.element(pop_code, c("HCB", "JPT"))])

prop <- snpgdsAdmixProp(RV, groups=groups)

# draw
plot(prop[, "YRI"], prop[, "CEU"], col=as.integer(tab$pop),
     xlab="Admixture Proportion from YRI",
     ylab="Admixture Proportion from CEU")
abline(v=0, col="gray25", lty=2)
abline(h=0, col="gray25", lty=2)
abline(a=-1, b=-1, col="gray25", lty=2)
legend("topright", legend=levels(tab$pop), pch="o", col=1:4)

# run eigen-analysis
RV <- snpgdsEIGMIX(genofile, sample.id=samp.id[pop_code=="JPT"])
z <- RV$ibdmat

mean(c(z))
mean(diag(z))
# close the genotype file
snpdgdsClose(genofile)

---

### snpgdsAlleleSwitch

**Allele-switching**

#### Description

Switch alleles according to the reference if needed.

#### Usage

```r
snpdgdsAlleleSwitch(gdsobj, A.allele, verbose=TRUE)
```

#### Arguments

- `gdsobj`: an object of class `SNPGDSFileClass`, a SNP GDS file
- `A.allele`: characters, referring to A allele
- `verbose`: if TRUE, show information

#### Value

A logical vector with TRUE indicating allele-switching and NA when it is unable to determine. NA occurs when `A.allele = NA` or `A.allele` is not in the list of alleles.

#### Author(s)

Xiuwen Zheng

#### Examples

```r
# the file name of SNP GDS
(fn <- snpgdsExampleFileName())

# copy the file
file.copy(fn, "test.gds", overwrite=TRUE)

# open the SNP GDS file
genofile <- snpgdsOpen("test.gds", readonly=FALSE)

# allelic information
allele <- read.gdsn(index.gdsn(genofile, "snp.allele"))
allele.list <- strsplit(allele, "/")
A.allele <- sapply(allele.list, function(x) { x[1] })
B.allele <- sapply(allele.list, function(x) { x[2] })

set.seed(1000)
flag <- rep(FALSE, length(A.allele))
flag[sample.int(length(A.allele), 50, replace=TRUE)] <- TRUE
A.allele[flag] <- B.allele[flag]
```
A.allele[sample.int(length(A.allele), 10, replace=TRUE)] <- NA
table(A.allele, exclude=NULL)

# allele switching
z <- snpgdsAlleleSwitch(genofile, A.allele)
table(z, exclude=NULL)

# close the file
snpgdsClose(genofile)

# delete the temporary file
unlink("test.gds", force=TRUE)

---

**snpdgsApartSelection**  
*Select SNPs with a basepair distance*

**Description**
Randomly selects SNPs for which each pair is at least as far apart as the specified basepair distance.

**Usage**

```r
snpdgsApartSelection(chromosome, position, min.dist=100000, 
                       max.n.snp.perchr=-1, verbose=TRUE)
```

**Arguments**
- `chromosome`: chromosome codes
- `position`: SNP positions in base pair
- `min.dist`: A numeric value to specify minimum distance required (in basepairs)
- `max.n.snp.perchr`: A numeric value specifying the maximum number of SNPs to return per chromosome, 
  "-1" means no number limit
- `verbose`: if TRUE, show information

**Value**
A logical vector indicating which SNPs were selected.

**Author(s)**
Xiuwen Zheng

**See Also**

`snpdgsLDpruning`
Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdsExampleFileName())
genofile

chr <- read.gdsn(index.gdsn(genofile, "snp.chromosome"))
pos <- read.gdsn(index.gdsn(genofile, "snp.position"))

set.seed(1000)
flag <- snpgdsApartSelection(chr, pos, min.dist=250000, verbose=TRUE)
table(flag)

# close the genotype file
snpdsClose(genofile)

snpdsBED2GDS  Conversion from PLINK BED to GDS

Description

Convert a PLINK binary ped file to a GDS file.

Usage

snpdsBED2GDS(bed.fn, fam.fn, bim.fn, out.gdsfn, family=FALSE,
snpfirstdim=NA, compress.annotation="ZIP_RA.max", compress.geno="",
option=NULL, cvt.chr=c("int", "char"), cvt.snpid=c("auto", "int"),
verbose=TRUE)

Arguments

bed.fn  the file name of binary file, genotype information
fam.fn  the file name of first six columns of ".ped"
bim.fn  the file name of extended MAP file: two extra columns = allele names
out.gdsfn  the output GDS file
family  if TRUE, to include family information in the sample annotation
snpfirstdim  if TRUE, genotypes are stored in the individual-major mode, (i.e. list all SNPs for the first individual, and then list all SNPs for the second individual, etc); NA, the dimension is determined by the BED file
compress.annotation  the compression method for the GDS variables, except "genotype"; optional values are defined in the function add.gdsn
compress.geno  the compression method for "genotype"; optional values are defined in the function add.gdsn
option  NULL or an object from snpgdsOption, see details
cvt.chr  "int" – chromosome code in the GDS file is integer; "char" – chromosome code in the GDS file is character
cvt.snpid  "int" – to create an integer snp.id starting from 1; "auto" – if SNP IDs in the PLINK file are not unique, to create an an integer snp.id, otherwise to use SNP IDs for snp.id
verbose  if TRUE, show information
Details

GDS – Genomic Data Structures, the extended file name used for storing genetic data, and the file format is used in the gdsfmt package.

BED – the PLINK binary ped format.

The user could use option to specify the range of code for autosomes. For humans there are 22 autosomes (from 1 to 22), but dogs have 38 autosomes. Note that the default settings are used for humans. The user could call option = snpgdsOption(autosome.end=38) for importing the BED file of dog. It also allow define new chromosome coding, e.g., option = snpgdsOption(Z=27).

Value

Return the file name of GDS format with an absolute path.

Author(s)

Xiuwen Zheng

References

Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, Maller J, Sklar P, de Bakker PIW, Daly MJ & Sham PC. 2007. PLINK: a toolset for whole-genome association and population-based linkage analysis. American Journal of Human Genetics, 81.

http://corearray.sourceforge.net/

See Also

snpgdsOption, snpgdsPED2GDS, snpgdsGDS2PED

Examples

# PLINK BED files
bed.fn <- system.file("extdata", "plinkhapmap.bed.gz", package="SNPRelate")
fam.fn <- system.file("extdata", "plinkhapmap.fam.gz", package="SNPRelate")
bim.fn <- system.file("extdata", "plinkhapmap.bim.gz", package="SNPRelate")

# convert
snpgdsBED2GDS(bed.fn, fam.fn, bim.fn, "HapMap.gds")

# open
genofile <- snpgdsOpen("HapMap.gds")
genofile

# close
snpgdsClose(genofile)

# delete the temporary file
unlink("HapMap.gds", force=TRUE)
**Description**

Close the SNP GDS file

**Usage**

```r
snpgdsClose(gdsobj)
```

**Arguments**

- `gdsobj` an object of class `SNPGDSFileClass`, a SNP GDS file

**Details**

It is suggested to call `snpgdsClose` instead of `closefn.gds`.

**Value**

None.

**Author(s)**

Xiuwen Zheng

**See Also**

- `snpgdsOpen`

**Examples**

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

genofile

# close the file
snpgdsClose(genofile)
```
snpgdsCombineGeno  

**Merge SNP datasets**

**Description**

To merge GDS files of SNP genotypes into a single GDS file

**Usage**

```r
snpgdsCombineGeno(gds.fn, out.fn, sample.id=NULL, snpobj=NULL, name.prefix=NULL, snpfirstdim=TRUE, compress.annotation="ZIP_RA.MAX", compress.geno=",", other.vars=NULL, verbose=TRUE)
```

**Arguments**

- `gds.fn`  
  a list of SNP GDS files to be merged
- `out.fn`  
  the name of output GDS file
- `sample.id`  
  NULL, or a list. If it is a list, specify sample ids for each SNP GDS file
- `snpobj`  
  specify a `snpgdsSNPListClass` object, used for strand switch; if NULL, the strand information of the first SNP GDS file is used
- `name.prefix`  
  NULL, a character vector (added to sample ids for each GDS file)
- `snpfirstdim`  
  if TRUE, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc)
- `compress.annotation`  
  the compression method for the variables except genotype
- `compress.geno`  
  the compression method for the variable genotype
- `other.vars`  
  a list object storing other variables
- `verbose`  
  if TRUE, show information

**Details**

The typical variables specified in `other.vars` are “sample.annot” and “snp.annot”, which are data.frame objects.

**Value**

None.

**Author(s)**

Xiuwen Zheng

**See Also**

`snpgdsCreateGeno`, `snpgdsCreateGenoSet`
### Examples

```r
# get the file name of a gds file
fn <- snpgdsExampleFileName()

# combine
snpgdsCombineGeno(c(fn, fn), "test.gds")

snpgdsSummary("test.gds")
```

### snpgdsCreateGeno

*Create a SNP genotype dataset from a matrix*

#### Description

To create a GDS file of genotypes from a matrix.

#### Usage

```r
snpgdsCreateGeno(gds.fn, genmat, sample.id=NULL, snp.id=NULL, snp.rs.id=NULL,
                  snp.chromosome=NULL, snp.position=NULL, snp.allele=NULL,
                  snpfirstdim=TRUE, compress.annotation="ZIP_RA.max", compress.geno="", other.vars=NULL)
```

#### Arguments

- **gds.fn**: the file name of gds
- **genmat**: a matrix of genotypes
- **sample.id**: the sample ids, which should be unique
- **snp.id**: the SNP ids, which should be unique
- **snp.rs.id**: the rs ids for SNPs, which can be not unique
- **snp.chromosome**: the chromosome indices
- **snp.position**: the SNP positions in basepair
- **snp.allele**: the reference/non-reference alleles
- **snpfirstdim**: if TRUE, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc)
- **compress.annotation**: the compression method for the variables except genotype
- **compress.geno**: the compression method for the variable genotype
- **other.vars**: a list object storing other variables

#### Details

There are possible values stored in the variable genmat: 0, 1, 2 and other values. “0” indicates two B alleles, “1” indicates one A allele and one B allele, “2” indicates two A alleles, and other values indicate a missing genotype.

If snpfirstdim is TRUE, then genmat should be “# of SNPs X # of samples”; if snpfirstdim is FALSE, then genmat should be “# of samples X # of SNPs”.

The typical variables specified in other.vars are “sample.annot” and “snp.annot”, which are data.frame objects.
**snpgdsCreateGenoSet**

Create a SNP genotype dataset from a GDS file

**Description**

To create a GDS file of genotypes from a specified GDS file.

**Usage**

```r
snpgdsCreateGenoSet(src.fn, dest.fn, sample.id=NULL, snp.id=NULL, snpfirstdim=NULL, compress.annotation="ZIP_RA.max", compress.geno="", verbose=TRUE)
```

**Arguments**

- `src.fn` the file name of a specified GDS file
- `dest.fn` the file name of output GDS file
- `sample.id` a vector of sample id specifying selected samples; if NULL, all samples are used
- `snp.id` a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- `snpfirstdim` if TRUE, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc)
- `compress.annotation` compression method for annotation
- `compress.geno` compression method for genotypes
- `verbose` if TRUE, print progress messages

**Examples**

```r
# load data
data(hapmap_geno)

# create a gds file
with(hapmap_geno, snpgdsCreateGeno("test.gds", genmat=genotype, 
sample.id=sample.id, snp.id=snp.id, snp.chromosome=snp.chromosome, 
snp.position=snp.position, snp.allele=snp.allele, snpfirstdim=TRUE))

# open the gds file
genofile <- snpgdsOpen("test.gds")

RV <- snpgdsPCA(genofile)
plot(RV$eigenvect[,2], RV$eigenvect[,1], xlab="PC 2", ylab="PC 1")

# close the file
snpgdsClose(genofile)
```
compress.annotation
the compression method for the variables except genotype
compress.geno
the compression method for the variable genotype
verbose
if TRUE, show information

Value
None.

Author(s)
Xiuwen Zheng

See Also

snpgdsCreateGenoSet, snpgdsCombineGeno

Examples

# open an example dataset (HapMap)
(genofile <- snpgdsOpen(snpgdsExampleFileName()))
# +  [ ] *
# |--- sample.id  { FStr8 279 ZIP(23.10%)}
# |--- snp.id  { Int32 9088 ZIP(34.76%)}
# |--- snp.rs.id  { FStr8 9088 ZIP(42.66%)}
# |--- snp.position  { Int32 9088 ZIP(94.73%)}
# |--- snp.chromosome  { UInt8 9088 ZIP(0.94%)} *
# |--- snp.allele  { FStr8 9088 ZIP(14.45%)}
# |--- genotype  { Bit2 9088x279 } *
# |--- sample.annot [ data.frame ] *
# | |--- sample.id  { FStr8 279 ZIP(23.10%)}
# | |--- family.id  { FStr8 279 ZIP(28.37%)}
# | |--- geneva.id  { Int32 9088 ZIP(80.29%)}
# | |--- father.id  { FStr8 279 ZIP(12.98%)}
# | |--- mother.id  { FStr8 279 ZIP(12.86%)}
# | |--- plate.id  { FStr8 279 ZIP(1.29%)}
# | |--- sex  { FStr8 279 ZIP(28.32%)}
# | |--- pop.group  { FStr8 279 ZIP(7.89%)}

set.seed(1000)
snpset <- unlist(snpgdsLDpruning(genofile))
length.snpset
# 6547

# close the file
snpgdsClose(genofile)
snpgdsCreateGenoSet(snpgdsExampleFileName(), "test.gds", snp.id=snpset)

########################################################################
# check

(gfile <- snpgdsOpen("test.gds"))
# +  [ ] *
# |--- sample.id  { VStr8 279 ZIP(29.89%)}
# |--- snp.id  { Int32 6547 ZIP(34.89%)}
snpGDS::snpGDSCutTree

# |--- snp.rs.id  { VStr8 6547 ZIP(40.52%)}
# |--- snp.position { Int32 6547 ZIP(94.85%)}
# |--- snp.chromosome { Int32 6547 ZIP(0.41%)}
# |--- snp.allele  { VStr8 6547 ZIP(11.51%)}
# |--- genotype  { Bit2 6547x279 } *

# close the file
snpGDS::snpGDSClose(gfile)

unlink("test.gds", force=TRUE)

---

snpGDS::snpGDSCutTree

**Determine clusters of individuals**

**Description**

To determine sub groups of individuals using a specified dendrogram from hierarchical cluster analysis

**Usage**

```r
snpGDS::snpGDSCutTree(hc, z.threshold=15, outlier.n=5, n.perm = 5000, samp.group=NULL, col.outlier="red", col.list=NULL, pch.outlier=4, pch.list=NULL, label.H=FALSE, label.Z=TRUE, verbose=TRUE)
```

**Arguments**

- `hc` an object of `snpGDS::snpGDSHCluster`
- `z.threshold` the threshold of Z score to determine whether split the node or not
- `outlier.n` the cluster with size less than or equal to `outlier.n` is considered as outliers
- `n.perm` the times for permutation
- `samp.group` if NULL, determine groups by Z score; if a vector of factor, assign each individual in dendrogram with respect to `samp.group`
- `col.outlier` the color of outlier
- `col.list` the list of colors for different clusters
- `pch.outlier` plotting 'character' for outliers
- `pch.list` plotting 'character' for different clusters
- `label.H` if TRUE, plotting heights in a dendrogram
- `label.Z` if TRUE, plotting Z scores in a dendrogram
- `verbose` if TRUE, show information

**Details**

The details will be described in future.
Value

Return a list:

- **sample.id**: the sample ids used in the analysis
- **z.threshold**: the threshold of Z score to determine whether split the node or not
- **outlier.n**: the cluster with size less than or equal to outlier.n is considered as outliers
- **samp.order**: the order of samples in the dendrogram
- **samp.group**: a vector of factor, indicating the group of each individual
- **dmatrix**: a matrix of pairwise group dissimilarity
- **dendrogram**: the dendrogram of individuals
- **merge**: a data.frame of (z, n1, n2) describing each combination: z, the Z score; n1, the size of the first cluster; n2, the size of the second cluster
- **clust.count**: the counts for clusters

Author(s)

Xiuwen Zheng

See Also

- `snpqdsHCluster`, `snpqdsDrawTree`, `snpqdsIBS`, `snpqdsDiss`

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpqdsExampleFileName())

pop.group <- as.factor(read.gdsn(index.gdsn(genofile, "sample.annot/pop.group")))

pop.level <- levels(pop.group)

diss <- snpgdsDiss(genofile)
hc <- snpgdsHCluster(diss)

# close the genotype file
snpqdsClose(genofile)
```

```
# cluster individuals
#
set.seed(100)
rv <- snpgdsCutTree(hc, label.H=TRUE, label.Z=TRUE)

# the distribution of Z scores
snpqdsDrawTree(rv, type="z-score", main="HapMap Phase II")

# draw dendrogram
snpqdsDrawTree(rv, main="HapMap Phase II",
edgePar=list(col=rgb(0.5,0.5,0.5, 0.75), t.col="black"))
```
### snpgdsDiss

**Individual dissimilarity analysis**

#### Description

Calculate the individual dissimilarities for each pair of individuals.

#### Usage

```r
snpgdsDiss(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE, remove.monosnp=TRUE, maf=NaN, missing.rate=NaN, num.thread=1, verbose=TRUE)
```

#### Arguments

- **gdsobj**: an object of class `SNPGDSFileClass`, a SNP GDS file
- **sample.id**: a vector of sample id specifying selected samples; if NULL, all samples are used
- **snp.id**: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
autosome.only  if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
remove.monosnp  if TRUE, remove monomorphic SNPs
maf  to use the SNPs with ">= maf" only; if NaN, no MAF threshold
missing.rate  to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
num.thread  the number of (CPU) cores used; if NA, detect the number of cores automatically
verbose  if TRUE, show information

Details
The minor allele frequency and missing rate for each SNP passed in snp.id are calculated over all the samples in sample.id.
The details will be described in future.

Value
Return a class "snpgdsDissClass":

sample.id  the sample ids used in the analysis
snp.id  the SNP ids used in the analysis
diss  a matrix of individual dissimilarity

Author(s)
Xiuwen Zheng

References


See Also
snpgdsHCluster

Examples
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

pop.group <- as.factor(read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))
pop.level <- levels(pop.group)

diss <- snpgdsDiss(genofile)
hc <- snpgdsHCluster(diss)

# close the genotype file
snpgdsClose(genofile)
# split
set.seed(100)
rv <- snpgdsCutTree(hc, label.H=TRUE, label.Z=TRUE)

# draw dendrogram
snpgdsDrawTree(rv, main="HapMap Phase II",
   edgePar=list(col=rgb(0.5,0.5,0.5, 0.75), t.col="black"))

---

**snpgdsDrawTree**  
*Draw a dendrogram*

**Description**

To draw a dendrogram or the distribution of Z scores

**Usage**

```
snpgdsDrawTree(obj, clust.count=NULL, dend.idx=NULL,  
type=c("dendrogram", "z-score"), yaxis.height=TRUE, yaxis.kinship=TRUE,  
y.kinship.baseline=NaN, y.label.kinship=FALSE, outlier.n=NULL,  
shadow.col=c(rgb(0.5, 0.5, 0.5, 0.25), rgb(0.5, 0.5, 0.5, 0.05)),  
outlier.col=rgb(1, 0.50, 0.50, 0.5), leaflab="none",  
labels=NULL, y.label=0.2, ...)
```

**Arguments**

- `obj`: an object returned by `snpgdsCutTree`
- `clust.count`: the counts for clusters, drawing shadows
- `dend.idx`: the index of sub tree, plot `obj$dendrogram[[dend.idx]]`, or NULL for the whole tree
- `type`: "dendrogram", draw a dendrogram; or "z-score", draw the distribution of Z score
- `yaxis.height`: if TRUE, draw the left Y axis: height of tree
- `yaxis.kinship`: if TRUE, draw the right Y axis: kinship coefficient
- `y.kinship.baseline`: the baseline value of kinship; if NaN, it is the height of the first split from top in a dendrogram; only works when `yaxis.kinship = TRUE`
- `y.label.kinship`: if TRUE, show 'PO/FS' etc on the right axis
- `outlier.n`: the cluster with size less than or equal to `outlier.n` is considered as outliers; if NULL, let `outlier.n = obj$outlier.n`
- `shadow.col`: two colors for shadow
- `outlier.col`: the colors for outliers
- `leaflab`: a string specifying how leaves are labeled. The default "perpendicular" write text vertically (by default). "textarea" writes text horizontally (in a rectangle), and "none" suppresses leaf labels.
- `labels`: the legend for different regions
- `y.label`: y positions of labels
- `...`: Arguments to be passed to the method "plot(, ...)", such as graphical parameters.
**snpgdsEIGMIX**

Eigen-analysis on SNP genotype data

**Description**

Eigen-analysis on IBD matrix based SNP genotypes.

**Usage**

```r
snpgdsEIGMIX(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE, remove.monosnp=TRUE, maf=NaN, missing.rate=NaN, num.thread=1L, eigen.cnt=32L, need.ibdmat=FALSE, ibdmat.only=FALSE, verbose=TRUE)
```

**Details**

The details will be described in future.

**Value**

None.

**Author(s)**

Xiuwen Zheng

**See Also**

[snpgdsCutTree](#)

**Examples**

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

pop.group <- as.factor(read.gdsn(index.gdsn(
    genofile, "sample.annot/pop.group")))
pop.level <- levels(pop.group)

diss <- snpgdsDiss(genofile)
hc <- snpgdsHCluster(diss)

# close the genotype file
snpgdsClose(genofile)

# split
set.seed(100)
rv <- snpgdsCutTree(hc, label.H=TRUE, label.Z=TRUE)

# draw dendrogram
snpgdsDrawTree(rv, main="HapMap Phase II",
               edgePar=list(col=rgb(0.5,0.5,0.5, 0.75), t.col="black"))
```
Arguments

- **gdsobj**: an object of class `SNPGDSFileClass`, a SNP GDS file.
- **sample.id**: a vector of sample id specifying selected samples; if NULL, all samples are used.
- **snp.id**: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used.
- **autosome.only**: if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome.
- **remove.mono.snp**: if TRUE, remove monomorphic SNPs.
- **maf**: to use the SNPs with ">= maf" only; if NaN, no MAF threshold.
- **missing.rate**: to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold.
- **num.thread**: the number of (CPU) cores used; if NA, detect the number of cores automatically.
- **eigen.cnt**: output the number of eigenvectors; if eigen.cnt <= 0, then return all eigenvectors.
- **need.ibdmat**: if TRUE, return the IBD matrix.
- **ibdmat.only**: return the IBD matrix only, do not compute the eigenvalues and eigenvectors.
- **verbose**: if TRUE, show information.

Value

Return a `snpgdsEIGMIXClass` object, and it is a list:

- **sample.id**: the sample ids used in the analysis.
- **snp.id**: the SNP ids used in the analysis.
- **eigenval**: eigenvalues.
- **eigenvect**: eigenvectors, "# of samples" x "eigen.cnt".
- **ibdmat**: the IBD matrix.

Author(s)

Xiuwen Zheng

References


See Also

`snpgdsEIGMIX`, `snpgdsPCA`

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen("snpgdsExampleFileName()")

# get population information
# or pop_code <- scan("pop.txt", what=character())
# if it is stored in a text file "pop.txt"
pop_code <- read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))

# get sample id
samp.id <- read.gdsn(index.gdsn(genofile, "sample.id"))
```
# run eigen-analysis
RV <- snpgdsEIGMIX(genofile)

# eigenvalues
RV$eigenval

# make a data.frame
tab <- data.frame(sample.id = samp.id, pop = factor(pop_code),
                  EV1 = RV$eigenvect[,1],       # the first eigenvector
                  EV2 = RV$eigenvect[,2],       # the second eigenvector
                  stringsAsFactors = FALSE)
head(tab)

# draw
plot(tab$EV2, tab$EV1, col=as.integer(tab$pop),
     xlab="eigenvector 2", ylab="eigenvector 1")
legend("topleft", legend=levels(tab$pop), pch="o", col=1:4)

# define groups
groups <- list(CEU = samp.id[pop_code == "CEU"],
                YRI = samp.id[pop_code == "YRI"],
                CHB = samp.id[is.element(pop_code, c("HCB", "JPT")))]
prop <- snpgdsAdmixProp(RV, groups=groups)

# draw
plot(prop[, "YRI"], prop[, "CEU"], col=as.integer(tab$pop),
     xlab = "Admixture Proportion from YRI",
     ylab = "Admixture Proportion from CEU")
abline(v=0, col="gray25", lty=2)
abline(h=0, col="gray25", lty=2)
abline(a=1, b=-1, col="gray25", lty=2)
legend("topright", legend=levels(tab$pop), pch="o", col=1:4)

# run eigen-analysis
RV <- snpgdsEIGMIX(genofile, sample.id=samp.id[pop_code=="JPT"],
                    need.ibdmat=TRUE)
z <- RV$ibdmat
mean(c(z))
mean(diag(z))

# close the genotype file
snpgdsClose(genofile)

---

**snpgdsErrMsg**

*Get the last error information*

**Description**

Return the last error message.
**Usage**

```
snpGdsErrMsg()
```

**Value**

Characters

**Author(s)**

Xiuwen Zheng

**Examples**

```
snpGdsErrMsg()
```

---

**snpGdsExampleFileName**  *Example GDS file*

**Description**

Return the file name of example data

**Usage**

```
snpGdsExampleFileName()
```

**Details**

A GDS genotype file was created from a subset of HapMap Phase II dataset consisting of 270 individuals and duplicates.

**Value**

Characters

**Author(s)**

Xiuwen Zheng

**Examples**

```
snpGdsExampleFileName()
```
Description

A SNPGDSFileClass object provides access to a GDS file containing genome-wide SNP data. It extends the class gds.class in the gdsfmt package.

Author(s)

Xiuwen Zheng

See Also

snpgdsOpen, snpgdsClose

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdgsExampleFileName())
genofile

class(genofile)
# "SNPGDSFileClass" "gds.class"

# close the file
snpgdsClose(genofile)

snpgdsFst

F-statistics (fixation indices)

Description

Calculate relatedness measures F-statistics (also known as fixation indices) for given populations

Usage

snpgdsFst(gdsobj, population, method=c("W&H02", "W&C84"), sample.id=NULL, snp.id=NULL, autosome.only=TRUE, remove.monosnp=TRUE, maf=NaN, missing.rate=NaN, with.id=FALSE, verbose=TRUE)

Arguments

gdsobj an object of class SNPGDSFileClass, a SNP GDS file
population a factor, indicating population information for each individual
sample.id a vector of sample id specifying selected samples; if NULL, all samples are used
snp.id a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
The minor allele frequency and missing rate for each SNP passed in `snp.id` are calculated over all the samples in `sample.id`.

### Value

Return a list:

- `sample.id`: the sample ids used in the analysis
- `snp.id`: the SNP ids used in the analysis
- `Fst`: Fst estimator
- `Beta`: Beta matrix

### Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

group <- as.factor(read.gdsn(index.gdsn(
    genofile, "sample.annot/pop.group")))

# Fst estimation
snpgdsFst(genofile, population=group, method="W&H02")

# or
snpgdsFst(genofile, population=group, method="W&C84")

# close the genotype file
snpgdsClose(genofile)
```
Description

Convert a GDS file to a PLINK binary ped file.

Usage

snpgdsGDS2BED(gdsobj, bed.fn, sample.id=NULL, snp.id=NULL, snpfirstdim=NULL, verbose=TRUE)

Arguments

gdsobj an object of class `SNPGDSFileClass`, a SNP GDS file; or characters, the file name of GDS
bed.fn the file name of output
sample.id a vector of sample id specifying selected samples; if NULL, all samples are used
snp.id a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
snpfirstdim if TRUE, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc); if NULL, determine automatically
verbose if TRUE, show information

Details

GDS – Genomic Data Structures, the extended file name used for storing genetic data, and the file format used in the gdsfmt package.
BED – the PLINK binary ped format.

Value

None.

Author(s)

Xiuwen Zheng

References

Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, Maller J, Sklar P, de Bakker PIW, Daly MJ & Sham PC. 2007. PLINK: a toolset for whole-genome association and population-based linkage analysis. American Journal of Human Genetics, 81.
http://corearray.sourceforge.net/

See Also

snpgdsBED2GDS, snpgdsGDS2PED
Example dataset (HapMap)

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

snpset <- snpgdsSelectSNP(genofile, missing.rate=0.95)
snpgdsGDS2BED(genofile, bed.fn="test", snp.id=snpset)

# close the genotype file
snpgdsClose(genofile)

# delete the temporary files
unlink(c("test.bed", "test.bim", "test.fam"), force=TRUE)
```

---

**snpgdsGDS2Eigen**  
*Conversion from GDS to Eigen (EIGENSTRAT)*

### Description

Convert a GDS file to an EIGENSTRAT file.

### Usage

```r
snpgdsGDS2Eigen(gdsobj, eigen.fn, sample.id=NULL, snp.id=NULL, verbose=TRUE)
```

### Arguments

- `gdsobj`  
an object of class `SNPGDSFileClass`, a SNP GDS file
- `eigen.fn`  
the file name of EIGENSTRAT
- `sample.id`  
a vector of sample id specifying selected samples; if NULL, all samples are used
- `snp.id`  
a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- `verbose`  
if TRUE, show information

### Details

GDS – Genomic Data Structures, the extended file name used for storing genetic data, and the file format used in the `gdsfmt` package.

Eigen – the text format used in EIGENSTRAT.

### Value

None.

### Author(s)

Xiuwen Zheng
References


http://corearray.sourceforge.net/

See Also

snpGdsGDS2PED

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpGdsExampleFileName())

snpset <- snpgdsSelectSNP(genofile, missing.rate=0.95)
snpGdsGDS2Eigen(genofile, eigen.fn="tmpeigen", snp.id=snpset)

# close the genotype file
snpGdsClose(genofile)

# delete the temporary files
unlink(c("tmpeigen.eigenstratgeno", "tmpeigen.ind", "tmpeigen.snp"), force=TRUE)

snpGdsGDS2PED  

Conversion from GDS to PED

Description

Convert a GDS file to a PLINK ped file.

Usage

snpGdsGDS2PED(gdsobj, ped.fn, sample.id=NULL, snp.id=NULL, use.snp.rsid=TRUE, format=c("A/G/C/T", "A/B", "1/2"), verbose=TRUE)

Arguments

- **gdsobj**: a GDS file object (`gds.class`)
- **ped.fn**: the file name of output
- **sample.id**: a vector of sample id specifying selected samples; if NULL, all samples are used
- **snp.id**: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- **use.snp.rsid**: if TRUE, use "snp.rs.id" instead of "snp.id" if available
- **format**: specify the coding: "A/G/C/T" – allelic codes stored in "snp.allele" of the GDS file; "A/B" – A and B codes; "1/2" – 1 and 2 codes
- **verbose**: if TRUE, show information
snpdgsGEN2GDS

Details
GDS – Genomic Data Structures, the extended file name used for storing genetic data, and the file format used in the gdsfmt package.
PED – the PLINK text ped format.

Value
None.

Author(s)
Xiuwen Zheng

References
Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, Maller J, Sklar P, de Bakker PIW, Daly MJ & Sham PC. 2007. PLINK: a toolset for whole-genome association and population-based linkage analysis. American Journal of Human Genetics, 81.
http://corearray.sourceforge.net/

See Also
snpdgsGDS2BED

Examples
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdgsExampleFileName())

# GDS ==> PED
snpdgsGDS2PED(genofile, ped.fn="tmp")

# close the GDS file
snpdgsClose(genofile)
Arguments

- **gen.fn**: the file name of Oxford GEN text file(s), it could be a vector indicate merging all files
- **sample.fn**: the file name of sample annotation
- **out.fn**: the output GDS file
- **chr.code**: a vector of chromosome code according to gen.fn, indicating chromosomes. It could be either numeric or character-type
- **call.threshold**: the threshold to determine missing genotypes
- **version**: either ">=2.0" or "<=1.1.5", see details
- **snpfirstdim**: if TRUE, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc)
- **compress.annotation**: the compression method for the GDS variables, except "genotype"; optional values are defined in the function add.gdsn
- **compress.geno**: the compression method for "genotype"; optional values are defined in the function add.gdsn
- **verbose**: if TRUE, show information

Details

GDS – Genomic Data Structures, the extended file name used for storing genetic data, and the file format is used in the gdsfmt package.

NOTE: the sample file format (sample.fn) has changed with the release of SNPTEST v2. Specifically, the way in which covariates and phenotypes are coded on the second line of the header file has changed. version has to be specified, and the function uses ">=2.0" by default.

Value

Return the file name of GDS format with an absolute path.

Author(s)

Xiuwen Zheng

References

http://www.stats.ox.ac.uk/~marchini/software/gwas/file_format.html

See Also

snpgdsBED2GDS, snpgdsVCF2GDS

Examples

cat("running snpgdsGEN2GDS ...\n")
## Not run:
snpgdsGEN2GDS("test.gen", "test.sample", "output.gds", chr.code=1)
## End(Not run)
snpgdsGetGeno

To get a genotype matrix

Description
To get a genotype matrix from a specified GDS file

Usage
snpgdsGetGeno(gdsobj, sample.id=NULL, snp.id=NULL, snpfirstdim=NA, .snpread=NA, with.id=FALSE, verbose=TRUE)

Arguments
gdsobj an object of class SNPGDSFileClass, a SNP GDS file; or characters to specify the file name of SNP GDS
sample.id a vector of sample id specifying selected samples; if NULL, all samples are used
snp.id a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
snpfirstdim if TRUE, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc); FALSE for snp-major mode; if NA, determine automatically
.snpread internal use
with.id if TRUE, return sample.id and snp.id
verbose if TRUE, show information

Value
The function returns an integer matrix with values 0, 1, 2 or NA representing the number of reference allele when with.id=FALSE; or list(genotype, sample.id, snp.id) when with.id=TRUE. The orders of sample and SNP IDs in the genotype matrix are actually consistent with sample.id and snp.id in the GDS file, which may not be as the same as the arguments sample.id and snp.id specified by users.

Author(s)
Xiuwen Zheng

Examples
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

set.seed(1000)
snpset <- sample(read.gdsn(index.gdsn(genofile, "snp.id")), 1000)

mat1 <- snpgdsGetGeno(genofile, snp.id=snpset, snpfirstdim=TRUE)
dim(mat1)
# 1000 279
table(c(mat1), exclude=NULL)

mat2 <- snpgdsGetGeno(genofile, snp.id=snpset, snpfirstdim=FALSE)
dim(mat2)
# 279 1000
table(c(mat2), exclude=NULL)

identical(t(mat1), mat2)
# TRUE

# close the file
snpgdsClose(genofile)

snpgdsGRM

**Genetic Relationship Matrix (GRM) for SNP genotype data**

**Description**

Calculate Genetic Relationship Matrix (GRM) using SNP genotype data.

**Usage**

```r
snpgdsGRM(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE, remove.monosnp=TRUE, maf=NaN, missing.rate=NaN, method=c("GCTA", "Eigenstrat", "EIGMIX", "W&Z15"), num.thread=1L, with.id=TRUE, verbose=TRUE)
```

**Arguments**

- **gdsobj** an object of class `SNPGDSFileClass`, a SNP GDS file
- **sample.id** a vector of sample id specifying selected samples; if NULL, all samples are used
- **snp.id** a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- **autosome.only** if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- **remove.monosnp** if TRUE, remove monomorphic SNPs
- **maf** to use the SNPs with ">= maf" only; if NaN, no MAF threshold
- **missing.rate** to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
- **num.thread** the number of (CPU) cores used; if NA, detect the number of cores automatically
- **with.id** if TRUE, the returned value with `sample.id` and `sample.id`
- **verbose** if TRUE, show information

**Value**

Return a list if `with.id = TRUE`:

- **sample.id** the sample ids used in the analysis
- **snp.id** the SNP ids used in the analysis
- **grm** the genetic relationship matrix; different methods might have different meanings and interpretation for estimates

If `with.id = FALSE`, this function returns the genetic relationship matrix (GRM) without sample and SNP IDs.
snpqdsHCluster

Author(s)

Xiuwen Zheng

References


See Also

snpqdsIndInb, snpqdsFst

Examples

# open an example dataset (HapMap)
genofile <- snpqdsOpen(snpqdsExampleFileName())

rv <- snpqdsGRM(genofile, method="GCTA")
eig <- eigen(rv$grm) # Eigen-decomposition

pop <- factor(read.gdsn(index.gdsn(genofile, "sample.annot/pop.group")))
plot(eig$vector[,1], eig$vector[,2], col=pop)
legend("topleft", legend=levels(pop), pch=19, col=1:4)

# close the file
snpqdsClose(genofile)

snpqdsHCluster Hierarchical cluster analysis

Description

Perform hierarchical cluster analysis on the dissimilarity matrix.

Usage

snpqdsHCluster(dist, sample.id=NULL, need.mat=TRUE, hang=0.25)
Arguments

- **dist**: an object of "snpgdsDissClass" from `snpgdsDiss`, an object of "snpgdsIBSClass" from `snpgdsIBS`, or a square matrix for dissimilarity.
- **sample.id**: to specify sample id, only work if dist is a matrix.
- **need.mat**: if TRUE, store the dissimilarity matrix in the result.
- **hang**: The fraction of the plot height by which labels should hang below the rest of the plot. A negative value will cause the labels to hang down from 0.

Details

Call the function `hclust` to perform hierarchical cluster analysis, using method="average".

Value

Return a list (class "snpgdsHCClass"):

- **sample.id**: the sample ids used in the analysis.
- **hclust**: an object returned from `hclust`.
- **dendrogram**: the dissimilarity matrix, if need.mat = TRUE.

Author(s)

Xiuwen Zheng

See Also

`snpgdsIBS`, `snpgdsDiss`, `snpgdsCutTree`

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

pop.group <- read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))
pop.group <- as.factor(pop.group)
pop.level <- levels(pop.group)

diss <- snpgdsDiss(genofile)
hc <- snpgdsHCluster(diss)
rv <- snpgdsCutTree(hc)
rv

# call 'plot' to draw a dendrogram
plot(rv$dendrogram, leaflab="none", main="HapMap Phase II")

# the distribution of Z scores
snpgdsDrawTree(rv, type="z-score", main="HapMap Phase II")

# draw dendrogram
snpgdsDrawTree(rv, main="HapMap Phase II",
               edgePar=list(col=rgb(0.5,0.5,0.5, 0.75), t.col="black"))
```
# close the file
snpgdsClose(genofile)

---

**snpgdsHWE**  
*Statistical test of Hardy-Weinberg Equilibrium*

**Description**

Calculate the p-values for the exact SNP test of Hardy-Weinberg Equilibrium.

**Usage**

`snpgdsHWE(gdsobj, sample.id=NULL, snp.id=NULL, with.id=FALSE)`

**Arguments**

- `gdsobj`: an object of class `SNPGDSFileClass`, a SNP GDS file
- `sample.id`: a vector of sample id specifying selected samples; if `NULL`, all samples will be used
- `snp.id`: a vector of snp id specifying selected SNPs; if `NULL`, all SNPs will be used
- `with.id`: if `TRUE`, the returned value with sample and SNP IDs

**Value**

If `with.id=FALSE`, return a vector of numeric values (p-value); otherwise, return a list with three components "pvalue", "sample.id" and "snp.id".

**Author(s)**

Xiuwen Zheng, Janis E. Wigginton

**References**


**See Also**

`snpgdsSNPRateFreq`

**Examples**

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

# Japanese samples
sample.id <- read.gdsn(index.gdsn(genofile, "sample.id"))
pop <- read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))
(samp.sel <- sample.id[pop=="JPT"])
samp.sel <- samp.sel[nchar(samp.sel) == 7]
```
# chromosome 1
snp.id <- snpgdsSelectSNP(genofile, sample.id=samp.sel, autosome.only=1L)

# HWE test
p <- snpgdsHWE(genofile, sample.id=samp.sel, snp.id=snp.id)
summary(p)

# QQ plot
plot(-log10((1:length(p))/length(p)), -log10(p[order(p)]),
xlab="-log10(expected P)", ylab="-log10(observed P)", main="QQ plot")
abline(a=0, b=1, col="blue")

# close the genotype file
snpgdsClose(genofile)

---

**snpgdsIBDKING**

*KING method of moment for the identity-by-descent (IBD) analysis*

**Description**

Calculate IBD coefficients by KING method of moment.

**Usage**

```r
snpgdsIBDKING(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE,
remove.monosnp=TRUE, maf=NaN, missing.rate=NaN,
type=c("KING-robust", "KING-homo"), family.id=NULL,
num.thread=1, verbose=TRUE)
```

**Arguments**

- `gdsobj`: an object of class `SNPGDSFileClass`, a SNP GDS file
- `sample.id`: a vector of sample id specifying selected samples; if NULL, all samples are used
- `snp.id`: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- `autosome.only`: if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- `remove.monosnp`: if TRUE, remove monomorphic SNPs
- `maf`: to use the SNPs with ">= maf" only; if NaN, no MAF threshold
- `missing.rate`: to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
- `type`: "KING-robust" – relationship inference in the presence of population stratification; "KING-homo" – relationship inference in a homogeneous population
- `family.id`: if NULL, all individuals are treated as singletons; if family id is given, within- and between-family relationship are estimated differently. If `sample.id=NULL, family.id` should have the same length as "sample.id" in the GDS file, otherwise `family.id` should have the same length and order as the argument `sample.id`
- `num.thread`: the number of (CPU) cores used; if NA, detect the number of cores automatically
- `verbose`: if TRUE, show information
Details

KING IBD estimator is a moment estimator, and it is computationally efficient relative to MLE method. The approaches include "KING-robust" – robust relationship inference within or across families in the presence of population substructure, and "KING-homo" – relationship inference in a homogeneous population.

With "KING-robust", the function would return the proportion of SNPs with zero IBS (IBS0) and kinship coefficient (kinship). With "KING-homo" it would return the probability of sharing one IBD (k1) and the probability of sharing zero IBD (k0).

The minor allele frequency and missing rate for each SNP passed in snp.id are calculated over all the samples in sample.id.

Value

Return a list:

- sample.id: the sample ids used in the analysis
- snp.id: the SNP ids used in the analysis
- k0: IBD coefficient, the probability of sharing zero IBD
- k1: IBD coefficient, the probability of sharing one IBD
- IBS0: proportion of SNPs with zero IBS
- kinship: the estimated kinship coefficients, if the parameter kinship=TRUE

Author(s)

Xiuwen Zheng

References


See Also

snpdgsIBDMLE, snpgdsIBDMoM

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdgsExampleFileName())

# CEU population
samp.id <- read.gdsn(index.gdsn(genofile, "sample.id"))
CEU.id <- samp.id[
  read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))=='CEU']

#### KING-robust:
#### relationship inference in the presence of population stratification
#### robust relationship inference across family

ibd.robust <- snpgdsIBDKING(genofile, sample.id=CEU.id, family.id=NULL)
```
names(ibd.robust)
# [1] "sample.id" "snp.id" "afreq" "IBS0" "kinship"

# select a set of pairs of individuals
dat <- snpgdsIBDSelection(ibd.robust, 1/32)
head(dat)

plot(dat$IBS0, dat$kinship, xlab="Proportion of Zero IBS",
     ylab="Estimated Kinship Coefficient (KING-robust)")

#### KING-robust:
#### relationship inference in the presence of population stratification
#### within- and between-family relationship inference

# incorporate with pedigree information
family.id <- read.gdsn(index.gdsn(genofile, "sample.annot/family.id"))
family.id <- family.id[match(CEU.id, samp.id)]

ibd.robust2 <- snpgdsIBDKING(genofile, sample.id=CEU.id, family.id=family.id)
names(ibd.robust2)

# select a set of pairs of individuals
dat <- snpgdsIBDSelection(ibd.robust2, 1/32)
head(dat)

plot(dat$IBS0, dat$kinship, xlab="Proportion of Zero IBS",
     ylab="Estimated Kinship Coefficient (KING-robust)")

#### KING-homo: relationship inference in a homogeneous population

ibd.homo <- snpgdsIBDKING(genofile, sample.id=CEU.id, type="KING-homo")
names(ibd.homo)
# "sample.id" "snp.id" "afreq" "k0" "k1"

# select a subset of pairs of individuals
dat <- snpgdsIBDSelection(ibd.homo, 1/32)
head(dat)

plot(dat$k0, dat$kinship, xlab="Pr(IBD=0)",
     ylab="Estimated Kinship Coefficient (KING-homo")

# close the genotype file
snpgdsClose(genofile)
Description

Calculate the three IBD coefficients (k0, k1, k2) for non-inbred individual pairs by Maximum Likelihood Estimation.

Usage

```r
snpgdsIBDMLE(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE,
remove.monosnp=TRUE, maf=NaN, missing.rate=NaN, kinship=FALSE,
kinship.constraint=FALSE, allele.freq=NULL,
method=c("EM", "downhill.simplex", "Jacquard"), max.niter=1000L,
reltol=sqrt(.Machine$double.eps), coeff.correct=TRUE,
out.num.iter=TRUE, num.thread=1, verbose=TRUE)
```

Arguments

- `gdsobj` an object of class `SNPGDSFileClass`, a SNP GDS file
- `sample.id` a vector of sample id specifying selected samples; if NULL, all samples are used
- `snp.id` a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- `autosome.only` if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- `remove.monosnp` if TRUE, remove monomorphic SNPs
- `maf` to use the SNPs with ">= maf" only; if NaN, no any MAF threshold
- `missing.rate` to use the SNPs with "<= missing.rate" only; if NaN, no any missing threshold
- `kinship` if TRUE, output the estimated kinship coefficients
- `kinship.constraint` if TRUE, constrict IBD coefficients ($k_0, k_1, k_2$) in the genelocial region ($2 k_0 k_1 >= k_2^2$)
- `allele.freq` to specify the allele frequencies; if NULL, determine the allele frequencies from gdsobj using the specified samples; if snp.id is specified, allele.freq should have the same order as snp.id
- `method` "EM", "downhill.simplex", "Jacquard"; see details
- `max.niter` the maximum number of iterations
- `reltol` relative convergence tolerance; the algorithm stops if it is unable to reduce the value of log likelihood by a factor of $\text{reltol} \times (\text{abs(log likelihood with the initial parameters)} + \text{reltol})$ at a step.
- `coeff.correct` TRUE by default, see details
- `out.num.iter` if TRUE, output the numbers of iterations
- `num.thread` the number of (CPU) cores used; if NA, detect the number of cores automatically
- `verbose` if TRUE, show information

Details

The minor allele frequency and missing rate for each SNP passed in snp.id are calculated over all the samples in sample.id.

The PLINK moment estimates are used as the initial values in the algorithm of searching maximum value of log likelihood function. Two numeric approaches can be used: one is Expectation-Maximization (EM) algorithm, and the other is Nelder-Mead method or downhill simplex method.
Generally, EM algorithm is more robust than downhill simplex method. "Jacquard" refers to the estimation of nine Jacquard’s coefficients.

If coeff.correct is TRUE, the final point that is found by searching algorithm (EM or downhill simplex) is used to compare the six points (fullsib, offspring, halfsib, cousin, unrelated), since any numeric approach might not reach the maximum position after a finite number of steps. If any of these six points has a higher value of log likelihood, the final point will be replaced by the best one.

Although MLE estimates are more reliable than MoM, MLE is much more computationally intensive than MoM, and might not be feasible to estimate pairwise relatedness for a large dataset.

Value

Return a snpgdsIBDClass object, which is a list:

- sample.id: the sample ids used in the analysis
- snp.id: the SNP ids used in the analysis
- afreq: the allele frequencies used in the analysis
- k0: IBD coefficient, the probability of sharing ZERO IBD, if method="EM" or "downhill.simplex"
- k1: IBD coefficient, the probability of sharing ONE IBD, if method="EM" or "downhill.simplex"
- D1, ..., D8: Jacquard’s coefficients, if method="Jacquard"
- kinship: the estimated kinship coefficients, if the parameter kinship=TRUE

Author(s)

Xiuwen Zheng

References


See Also

snpgdsIBDMELogLik, snpgdsIBDMoM

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpExampleFileName())

YRI.id <- read.gdsn(index.gdsn(genofile, "sample.id"))[read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))="YRI"]
YRI.id <- YRI.id[1:30]

# SNP pruning
set.seed(10)
snpset <- snpgdsLDpruning(genofile, sample.id=YRI.id, maf=0.05, missing.rate=0.05)
snpset <- sample(unlist(snpset), 250)
mibd <- snpgdsIBDMLE(genofile, sample.id=YRI.id, snp.id=snpset)

# select a set of pairs of individuals
d <- snpgdsIBDSelection(mibd, kinship.cutoff=1/8)
head(d)

# log likelihood
loglik <- snpgdsIBDMLELogLik(genofile, mibd)
loglik0 <- snpgdsIBDMLELogLik(genofile, mibd, relatedness="unrelated")

# likelihood ratio test
p.value <- pchisq(loglik - loglik0, 1, lower.tail=FALSE)

flag <- lower.tri(mibd$k0)
plot(NaN, xlim=c(0,1), ylim=c(0,1), xlab="k0", ylab="k1")
lines(c(0,1), c(1,0), col="red", lty=3)
points(mibd$k0[flag], mibd$k1[flag])

# specify the allele frequencies
afreq <- snpgdsSNPRateFreq(genofile, sample.id=YRI.id, snp.id=snpset)$AlleleFreq
subibd <- snpgdsIBDMLE(genofile, sample.id=YRI.id[1:25], snp.id=snpset, allele.freq=afreq)

summary(c(subibd$k0 - mibd$k0[1:25, 1:25]))
# ZERO
summary(c(subibd$k1 - mibd$k1[1:25, 1:25]))
# ZERO

# close the genotype file
snpgdsClose(genofile)

---

**snpgdsIBDMLELogLik**

Log likelihood for MLE method in the Identity-By-Descent (IBD) Analysis

**Description**

Calculate the log likelihood values from maximum likelihood estimation.

**Usage**

```
snpgdsIBDMLELogLik(gdsobj, ibdobj, k0 = NaN, k1 = NaN,
relatedness=c("", "self", "fullsib", "offspring",
"halfsib", "cousin", "unrelated"))
```
Arguments

- **gdsobj**: an object of class `SNPGDSFileClass`, a SNP GDS file
- **ibdobj**: the `snpgdsIBDClass` object returned from `snpgdsIBDMLE`
- **k0**: specified IBD coefficient
- **k1**: specified IBD coefficient
- **relatedness**: specify a relatedness, otherwise use the values of k0 and k1

Details

If `(relatedness == "")` and `(k0 == NaN or k1 == NaN)`, then return the log likelihood values for each `(k0, k1)` stored in ibdobj. If `(relatedness == "")` and `(k0 != NaN) and (k1 != NaN)`, then return the log likelihood values for a specific IBD coefficient `(k0, k1)`. If `relatedness` is: "self", then `k0 = 0`, `k1 = 0"; "fullsib", then `k0 = 0.25, k1 = 0.5; "offspring", then `k0 = 0, k1 = 1; "halfsib", then `k0 = 0.5, k1 = 0.5; "cousin", then `k0 = 0.75, k1 = 0.25; "unrelated", then `k0 = 1, k1 = 0.

Value

Return a n-by-n matrix of log likelihood values, where n is the number of samples.

Author(s)

- Xiuwen Zheng

References


See Also

- `snpgdsIBDMLE`
- `snpgdsIBDMoM`

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpGdsExampleFileName())

YRI.id <- read.gdsn(index.gdsn(genofile, "sample.id"))[
  read.gdsn(index.gdsn(genofile, "sample.annot/pop.group")) == "YRI"]
YRI.id <- YRI.id[1:30]

# SNP pruning
set.seed(10)
snpset <- snpgdsLDpruning(genofile, sample.id=YRI.id, maf=0.05, missing.rate=0.05)
snpset <- sample(unlist(snpset), 250)
mibd <- snpgdsIBDMLE(genofile, sample.id=YRI.id, snp.id=snpset)
names(mibd)

# select a set of pairs of individuals
```
```r
d <- snpgdsIBDSelection(mibd, kinship.cutoff=1/8)
head(d)

# log likelihood
loglik <- snpgdsIBDMLELogLik(geno, mibd)
loglik0 <- snpgdsIBDMLELogLik(geno, mibd, relatedness="unrelated")

# likelihood ratio test
p.value <- pchisq(loglik - loglik0, 1, lower.tail=FALSE)

flag <- lower.tri(mibd$k0)
plot(NaN, xlim=c(0,1), ylim=c(0,1), xlab="k0", ylab="k1")
lines(c(0,1), c(1,0), col="red", lty=3)
points(mibd$k0[flag], mibd$k1[flag])

# specify the allele frequencies
afreq <- snpgdsSNPRateFreq(geno, sample.id=YRI.id, 
snp.id=snpset)$AlleleFreq
subibd <- snpgdsIBDMLE(geno, sample.id=YRI.id[1:25], snp.id=snpset, 
allele.freq=afreq)
summary(c(subibd$k0 - mibd$k0[1:25, 1:25]))
# ZERO
summary(c(subibd$k1 - mibd$k1[1:25, 1:25]))
# ZERO

# close the genotype file
snpgdsClose(geno)
```

---

**snpgdsIBDMoM**

**PLINK method of moment (MoM) for the Identity-By-Descent (IBD) Analysis**

**Description**

Calculate three IBD coefficients for non-inbred individual pairs by PLINK method of moment (MoM).

**Usage**

```r
snpgdsIBDMoM(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE, 
remove.monosnp=TRUE, maf=NaN, missing.rate=NaN, allele.freq=NULL, 
kinship=FALSE, kinship.constraint=FALSE, num.thread=1, verbose=TRUE)
```

**Arguments**

- `gdsobj` an object of class **SNPGDSFileClass**, a SNP GDS file
- `sample.id` a vector of sample id specifying selected samples; if NULL, all samples are used
- `snp.id` a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
autosome.only if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
remove.monosnp if TRUE, remove monomorphic SNPs
maf to use the SNPs with "\( \geq \) maf" only; if NaN, no MAF threshold
missing.rate to use the SNPs with "\( \leq \) missing.rate" only; if NaN, no missing threshold
allele.freq to specify the allele frequencies; if NULL, determine the allele frequencies from gdsobj using the specified samples; if snp.id is specified, allele.freq should have the same order as snp.id
kinship if TRUE, output the estimated kinship coefficients
kinship.constraint if TRUE, constrict IBD coefficients \( \{k_0, k_1, k_2\} \) in the genetical region \( \{2k_0k_1 \geq k_2^2\} \)
num.thread the number of (CPU) cores used; if NA, detect the number of cores automatically
verbose if TRUE, show information

Details

PLINK IBD estimator is a moment estimator, and it is computationally efficient relative to MLE method. In the PLINK method of moment, a correction factor based on allele counts is used to adjust for sampling. However, if allele frequencies are specified, no correction factor is conducted since the specified allele frequencies are assumed to be known without sampling.

The minor allele frequency and missing rate for each SNP passed in snp.id are calculated over all the samples in sample.id.

Value

Return a list:

- sample.id the sample ids used in the analysis
- snp.id the SNP ids used in the analysis
- k0 IBD coefficient, the probability of sharing ZERO IBD
- k1 IBD coefficient, the probability of sharing ONE IBD
- kinship the estimated kinship coefficients, if the parameter kinship=TRUE

Author(s)

Xiuwen Zheng

References

Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, Maller J, Sklar P, de Bakker PIW, Daly MJ & Sham PC. 2007. PLINK: a toolset for whole-genome association and population-based linkage analysis. American Journal of Human Genetics, 81.

See Also

snpdgsIBDMLE, snpgdsIBDMLELogLik
Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdgsExampleFileName())

#########################################################
# CEU population
CEU.id <- read.gdsn(index.gdsn(genofile, "sample.id"))[
  read.gdsn(index.gdsn(genofile, "sample.annot/pop.group")) == "CEU"
]  
pibd <- snpgdsIBDMoM(genofile, sample.id=CEU.id)

flag <- lower.tri(pibd$k0)

plot(NaN, xlim=c(0,1), ylim=c(0,1), xlab="k0", ylab="k1")
lines(c(0,1), c(1,0), col="red", lty=3)
points(pibd$k0[flag], pibd$k1[flag])

# select a set of pairs of individuals
d <- snpgdsIBDSelection(pibd, kinship.cutoff=1/8)
head(d)

#########################################################
# YRI population
YRI.id <- read.gdsn(index.gdsn(genofile, "sample.id"))[
  read.gdsn(index.gdsn(genofile, "sample.annot/pop.group")) == "YRI"
]  
pibd <- snpgdsIBDMoM(genofile, sample.id=YRI.id)

flag <- lower.tri(pibd$k0)

plot(NaN, xlim=c(0,1), ylim=c(0,1), xlab="k0", ylab="k1")
lines(c(0,1), c(1,0), col="red", lty=3)
points(pibd$k0[flag], pibd$k1[flag])

# specify the allele frequencies
afreq <- snpgdsSNPRateFreq(genofile, sample.id=YRI.id)$AlleleFreq
aibd <- snpgdsIBDMoM(genofile, sample.id=YRI.id, allele.freq=afreq)

flag <- lower.tri(aibd$k0)

plot(NaN, xlim=c(0,1), ylim=c(0,1), xlab="k0", ylab="k1")
lines(c(0,1), c(1,0), col="red", lty=3)
points(aibd$k0[flag], aibd$k1[flag])

# analysis on a subset
subibd <- snpgdsIBDMoM(genofile, sample.id=YRI.id[1:25], allele.freq=afreq)

summary(c(subibd$k0 - aibd$k0[1:25, 1:25]))
# ZERO
summary(c(subibd$k1 - aibd$k1[1:25, 1:25]))
# ZERO

# close the genotype file
snpgdsClose(genofile)

snpdgsIBDSelection  Get a table of IBD coefficients
Description
Return a data frame with IBD coefficients.

Usage
snpgdsIBDSelection(ibdobj, kinship.cutoff=NaN, samp.sel=NULL)

Arguments
ibdobj          an object of snpgdsIBDClass returned by snpgdsIBDMLE or snpgdsIBDMoM
kinship.cutoff  select the individual pairs with kinship coefficients >= kinship.cutoff; no filter if kinship.cutoff = NaN
samp.sel        a logical vector or integer vector to specify selection of samples

Value
Return a data.frame:

ID1           the id of the first individual
ID2           the id of the second individual
k0            the probability of sharing ZERO alleles
k1            the probability of sharing ONE alleles
kinship       kinship coefficient

Author(s)
Xiuwen Zheng

See Also
snpgdsIBDMLE, snpgdsIBDMoM, snpgdsIBDKING

Examples
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

# YRI population
YRI.id <- read.gdsn(index.gdsn(genofile, "sample.id"))[read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))=='YRI']
pibd <- snpgdsIBDMoM(genofile, sample.id=YRI.id)
flag <- lower.tri(pibd$k0)
plot(NaN, xlim=c(0,1), ylim=c(0,1), xlab="k0", ylab="k1")
lines(c(0,1), c(1,0), col="red", lty=3)
points(pibd$k0[flag], pibd$k1[flag])

# close the genotype file
snpgdsClose(genofile)

# IBD coefficients
dat <- snpgdsIBDSelection(pibd, 1/32)
head(dat)
#     ID1  ID2   k0   k1 kinship
# snpgdsIBS

## snpgdsIBS

### Identity-By-State (IBS) proportion

#### Description

Calculate the fraction of identity by state for each pair of samples

#### Usage

```r
snpgdsIBS(gdsobj, sample.id = NULL, snp.id = NULL, autosome.only = TRUE, remove.monosnp = TRUE, maf = NaN, missing.rate = NaN, num.thread = 1L, verbose = TRUE)
```

#### Arguments

- **gdsobj**: an object of class `SNPGDSFileClass`, a SNP GDS file
- **sample.id**: a vector of sample id specifying selected samples; if NULL, all samples are used
- **snp.id**: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- **autosome.only**: if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- **remove.monosnp**: if TRUE, remove monomorphic SNPs
- **maf**: to use the SNPs with ">= maf" only; if NaN, no MAF threshold
- **missing.rate**: to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
- **num.thread**: the number of (CPU) cores used; if NA, detect the number of cores automatically
- **verbose**: if TRUE, show information

#### Details

The minor allele frequency and missing rate for each SNP passed in `snp.id` are calculated over all the samples in `sample.id`.

The values of the IBS matrix range from ZERO to ONE.

#### Value

Return a list (class "snpgdsIBSClасс"):

- **sample.id**: the sample ids used in the analysis
- **snp.id**: the SNP ids used in the analysis
- **ibs**: a matrix of IBS proportion, "># of samples" x "># of samples"
snpgdsIBSNum

Identity-By-State (IBS)

Description

Calculate the number of SNPs for identity by state for each pair of samples.

Usage

snpgdsIBSNum(gdsobj, sample.id = NULL, snp.id = NULL, autosome.only = TRUE, remove.monosnp = TRUE, maf = NaN, missing.rate = NaN, num.thread = 1L, verbose = TRUE)

Arguments

- gdsobj: an object of class `SNPGDSFileClass`, a SNP GDS file
- sample.id: a vector of sample id specifying selected samples; if NULL, all samples are used
- snp.id: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- autosome.only: if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- remove.monosnp: if TRUE, remove monomorphic SNPs
- maf: to use the SNPs with ">= maf" only; if NaN, no MAF threshold
- missing.rate: to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
- num.thread: the number of (CPU) cores used; if NA, detect the number of cores automatically
- verbose: if TRUE, show information

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

# perform identity-by-state calculations
ibs <- snpgdsIBS(genofile)

# perform multidimensional scaling analysis on
# the genome-wide IBS pairwise distances:
loc <- cmdscale(1 - ibs$ibs, k = 2)
x <- loc[, 1]; y <- loc[, 2]
race <- as.factor(read.gdsn(index.gdsn(genofile, "sample.annot/pop.group")))
plot(x, y, col=race, xlab = "", ylab = "", main = "cmdscale(IBS Distance)"
legend("topleft", legend=levels(race), text.col=1:nlevels(race))

# close the file
snpgdsClose(genofile)
Details

The minor allele frequency and missing rate for each SNP passed in snp.id are calculated over all the samples in sample.id.

Value

Return a list (n is the number of samples):

- sample.id: the sample ids used in the analysis
- snp.id: the SNP ids used in the analysis
- ibs0: a n-by-n matrix, the number of SNPs sharing 0 IBS
- ibs1: a n-by-n matrix, the number of SNPs sharing 1 IBS
- ibs2: a n-by-n matrix, the number of SNPs sharing 2 IBS

Author(s)

Xiuwen Zheng

See Also

- snpgdsIBS

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdfsExampleFileName())

RV <- snpgdsIBSNum(genofile)
pop <- read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))
L <- order(pop)
image(RVibs0[L, L]/length(RV$snp.id))

# close the genotype file
snpgdsClose(genofile)
```

---

**snpdfsIndInb**

*Individual Inbreeding Coefficients*

Description

To calculate individual inbreeding coefficients using SNP genotype data

Usage

```r
snpdfsIndInb(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE, remove.monosnp=TRUE, maf=NaN, missing.rate=NaN, method=c("mom.weir", "mom.visscher", "mle"), allele.freq=NULL, out.num.iter=TRUE, reltol=.Machine$double.eps^0.75, verbose=TRUE)
```
Arguments

- gdsobj: an object of class \texttt{SNPGDSClass}, a SNP GDS file
- sample.id: a vector of sample id specifying selected samples; if NULL, all samples are used
- snp.id: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- autosome.only: if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- remove.monosnp: if TRUE, remove monomorphic SNPs
- maf: to use the SNPs with ">= maf" only; if NaN, no MAF threshold
- missing.rate: to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
- method: see details
- allele.freq: to specify the allele frequencies; if NULL, the allele frequencies are estimated from the given samples
- out.num.iter: output the numbers of iterations
- reltol: relative convergence tolerance used in MLE; the algorithm stops if it is unable to reduce the value of log likelihood by a factor of \$reltol * (abs(log likelihood with the initial parameters) + reltol)\$ at a step.
- verbose: if TRUE, show information

Details

The method can be: "mom.weir": a modified Visscher’s estimator, proposed by Bruce Weir; "mom.visscher": Visscher’s estimator described in Yang et al. (2010); "mle": the maximum likelihood estimation.

Value

Return estimated inbreeding coefficient.

Author(s)

Xiuwen Zheng

References


Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

rv <- snpgdsIndInb(genofile, method="mom.visscher")
head(rv$inbreeding)
summary(rv$inbreeding)

# close the genotype file
snpgdsClose(genofile)
```
Description

To calculate an individual inbreeding coefficient using SNP genotype data.

Usage

```r
snpdgsIndInbCoef(x, p, method = c("mom.weir", "mom.visscher", "mle"), reltol=.Machine$double.eps^0.75)
```

Arguments

- `x`: SNP genotypes
- `p`: allele frequencies
- `method`: see details
- `reitol`: relative convergence tolerance used in MLE; the algorithm stops if it is unable to reduce the value of log likelihood by a factor of `reitol` * (`abs(log likelihood with the initial parameters) + reitol`) at a step.

Details

The method can be: "mom.weir": a modified Visscher’s estimator, proposed by Bruce Weir; "mom.visscher": Visscher’s estimator described in Yang et al. (2010); "mle": the maximum likelihood estimation.

Value

Return estimated inbreeding coefficient.

Author(s)

Xiuwen Zheng

References


Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdgsExampleFileName())

chr1 <- read.gdsn(index.gdsn(genofile, "snp.id"))[
  read.gdsn(index.gdsn(genofile, "snp.chromosome")) == 1]
chr1idx <- match(chr1, read.gdsn(index.gdsn(genofile, "snp.id")))

AF <- snpgdsSNPRateFreq(genofile)
g <- read.gdsn(index.gdsn(genofile, "genotype"), start=c(1,1), count=c(-1,1))
```
Individual inbreeding and relatedness estimation (beta estimator)

Description

Calculate individual inbreeding and relatedness estimation (beta estimator) using SNP genotype data.

Usage

```
snpgdsIndivBeta(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE,
remove.monosnp=TRUE, maf=NaN, missing.rate=NaN,
method=c("weighted"), num.thread=1L, with.id=TRUE, verbose=TRUE)
```

Arguments

- **gdsobj**: an object of class `SNPGDSFileClass`, a SNP GDS file
- **sample.id**: a vector of sample id specifying selected samples; if NULL, all samples are used
- **snp.id**: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- **autosome.only**: if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- **remove.monosnp**: if TRUE, remove monomorphic SNPs
- **maf**: to use the SNPs with ">= maf" only; if NaN, no MAF threshold
- **missing.rate**: to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
- **method**: "weighted" estimator
- **num.thread**: the number of (CPU) cores used; if NA, detect the number of cores automatically
- **with.id**: if TRUE, the returned value with sample.id and sample.id
- **verbose**: if TRUE, show information

Value

Return a list if with.id = TRUE:

- **sample.id**: the sample ids used in the analysis
- **snp.id**: the SNP ids used in the analysis
- **beta**: the genetic relationship matrix; different methods might have different meanings and interpretation for estimates

If with.id = FALSE, this function returns the genetic relationship matrix without sample and SNP IDs.
snpgdsLDMat

Author(s)
Xiuwen Zheng

References

See Also
snpgdsGRM, snpgdsIndInb, snpgdsFst

Examples
library(SNPRelate)
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

rv <- snpgdsGRM(genofile, method="W&Z15")

beta <- snpgdsIndivBeta(genofile, with.id=FALSE)
beta[1:10, 1:10]

# close the file
snpgdsClose(genofile)

snpgdsLDMat     Linkage Disequilibrium (LD) analysis

Description
Return a LD matrix for SNP pairs.

Usage
snpgdsLDMat(gdsobj, sample.id=NULL, snp.id=NULL, slide=250L,
method=c("composite", "r", "dprime", "corr", "cov"), mat.trim=FALSE,
num.thread=1L, with.id=TRUE, verbose=TRUE)

Arguments
gdsobj       an object of class SNPGDSFileClass, a SNP GDS file
sample.id    a vector of sample id specifying selected samples; if NULL, all samples are used
snp.id       a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
slide        # of SNPs, the size of sliding window, see details
method       "composite", "r", "dprime", "corr", "cov", see details
mat.trim     if TRUE, trim the matrix when slide > 0: the function returns a "num_slide x (n.snp - slide)" matrix
num.thread   the number of (CPU) cores used; if NA, detect the number of cores automatically
with.id      if TRUE, the returned value with sample.id and sample.id
verbose      if TRUE, show information
Details

Four methods can be used to calculate linkage disequilibrium values: "composite" for LD composite measure, "r" for R coefficient (by EM algorithm assuming HWE, it could be negative), "dprime" for D', and "corr" for correlation coefficient. The method "corr" is equivalent to "composite", when SNP genotypes are coded as: 0 – BB, 1 – AB, 2 – AA.

If slide \( \leq 0 \), the function returns a n-by-n LD matrix where the value of i row and j column is LD of i and j SNPs. If slide \( > 0 \), it returns a m-by-n LD matrix where n is the number of SNPs, m is the size of sliding window, and the value of i row and j column is LD of j and j+i SNPs.

Value

Return a list:

- sample.id: the sample ids used in the analysis
- snp.id: the SNP ids used in the analysis
- LD: a matrix of LD values
- slide: the size of sliding window

Author(s)

Xiuwen Zheng

References


See Also

snpgdsLDpair, snpgdsLDpruning

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdgdsExampleFileName())

# chromosome 15
snpset <- read.gdsn(index.gdsn(genofile, "snp.id"))[
  read.gdsn(index.gdsn(genofile, "snp.chromosome")) == 15]

# LD matrix without sliding window
ld.noslide <- snpgdsLDMat(genofile, snp.id=snpset, slide=-1, method="composite")
# plot
image(t(ld.noslide$LD^2), col=terrain.colors(16))

# LD matrix with a sliding window
ld.slide <- snpgdsLDMat(genofile, snp.id=snpset, method="composite")
# plot
image(t(ld.slide$LD^2), col=terrain.colors(16))
```
# close the genotype file
snpgdsClose(genofile)

## snpgdsLDpair

**Linkage Disequilibrium (LD)**

### Description
Return a LD value between snp1 and snp2.

### Usage

```r
snpgdsLDpair(snp1, snp2, method = c("composite", "r", "dprime", "corr"))
```

### Arguments

- **snp1**: a vector of SNP genotypes (0 – BB, 1 – AB, 2 – AA)
- **snp2**: a vector of SNP genotypes (0 – BB, 1 – AB, 2 – AA)
- **method**: "composite", "r", "dprime", "corr", see details

### Details
Four methods can be used to calculate linkage disequilibrium values: "composite" for LD composite measure, "r" for R coefficient (by EM algorithm assuming HWE, it could be negative), "dprime" for D', and "corr" for correlation coefficient. The method "corr" is equivalent to "composite", when SNP genotypes are coded as: 0 – BB, 1 – AB, 2 – AA.

### Value
Return a numeric vector:

- **ld**: a measure of linkage disequilibrium
- **pA_A**: haplotype frequency of AA, the first locus is A and the second locus is A
- **pA_B**: haplotype frequency of AB, the first locus is A and the second locus is B
- **pB_A**: haplotype frequency of BA, the first locus is B and the second locus is A
- **pB_B**: haplotype frequency of BB, the first locus is B and the second locus is B

### Author(s)
Xiuwen Zheng

### References

snpgdsLDpruning

**Description**

Recursively removes SNPs within a sliding window

**Usage**

```r
snpgdsLDpruning(gdsobj, sample.id = NULL, snp.id = NULL, autosome.only = TRUE,
remove.monosnp = TRUE, maf = NaN, missing.rate = NaN,
method = c("composite", "r", "dprime", "corr"), slide.max.bp = 500000,
slide.max.n = NA, ld.threshold = 0.2, num.thread = 1, verbose = TRUE)
```

**Arguments**

- **gdsobj**: an object of class `SNPGDSFileClass`, a SNP GDS file
- **sample.id**: a vector of sample id specifying selected samples; if NULL, all samples are used
- **snp.id**: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- **autosome.only**: if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- **remove.monosnp**: if TRUE, remove monomorphic SNPs
- **maf**: to use the SNPs with ">= maf" only; if NaN, no MAF threshold
- **missing.rate**: to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
- **method**: "composite", "r", "dprime", "corr", see details
- **slide.max.bp**: the maximum basepairs in the sliding window
- **slide.max.n**: the maximum number of SNPs in the sliding window
- **ld.threshold**: the LD threshold
- **num.thread**: the number of (CPU) cores used; if NA, detect the number of cores automatically
- **verbose**: if TRUE, show information

**Examples**

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())
snp1 <- read.gdsn(index.gdsn(genofile, "genotype"), start=c(1,1), count=c(1,-1))
snp2 <- read.gdsn(index.gdsn(genofile, "genotype"), start=c(2,1), count=c(1,-1))

snpgdsLDpair(snp1, snp2, method = "composite")
snpgdsLDpair(snp1, snp2, method = "r")
snpgdsLDpair(snp1, snp2, method = "dprime")
snpgdsLDpair(snp1, snp2, method = "corr")

# close the genotype file
snpgdsClose(genofile)
```

**See Also**

`snpgdsLDMat`, `snpgdsLDpruning`
**Details**

The minor allele frequency and missing rate for each SNP passed in `snp.id` are calculated over all the samples in `sample.id`.

Four methods can be used to calculate linkage disequilibrium values: "composite" for LD composite measure, "r" for R coefficient (by EM algorithm assuming HWE, it could be negative), "dprime" for D', and "corr" for correlation coefficient. The method "corr" is equivalent to "composite", when SNP genotypes are coded as: 0 – BB, 1 – AB, 2 – AA. The argument `ld.threshold` is the absolute value of measurement.

It is useful to generate a pruned subset of SNPs that are in approximate linkage equilibrium with each other. The function `snpgdsLDpruning` recursively removes SNPs within a sliding window based on the pairwise genotypic correlation. SNP pruning is conducted chromosome by chromosome, since SNPs in a chromosome can be considered to be independent with the other chromosomes.

The pruning algorithm on a chromosome is described as follows (n is the total number of SNPs on that chromosome):

1) Randomly select a starting position i, and let the current SNP set $S = \{i\}$;
2) For each right position j from i+1 to n: if any LD between j and k is greater than `ld.threshold`, where k belongs to S, and both of j and k are in the sliding window, then skip j; otherwise, let S be $S + \{j\}$;
3) For each left position j from i-1 to 1: if any LD between j and k is greater than `ld.threshold`, where k belongs to S, and both of j and k are in the sliding window, then skip j; otherwise, let S be $S + \{j\}$;
4) Output S, the final selection of SNPs.

**Value**

Return a list of SNP IDs stratified by chromosomes.

**Author(s)**

Xiuwen Zheng

**References**


**See Also**

`snpgdsLDMat`, `snpgdsLDpair`

**Examples**

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())
set.seed(1000)
snpset <- snpgdsLDpruning(genofile)
names(snpset)
```
Open a SNP GDS File

Description
Open a SNP GDS file

Usage
snpdgsOpen(filename, readonly=TRUE, allow.duplicate=FALSE, allow.fork=FALSE)

Arguments
filename the file name
readonly whether read-only or not
allow.duplicate if TRUE, it is allowed to open a GDS file with read-only mode when it has been opened in the same R session, see openfn.gds
allow.fork TRUE for parallel environment using forking, see openfn.gds

Details
It is strongly suggested to call snpgdsOpen instead of openfn.gds, since snpgdsOpen will perform internal checking for data integrity.

Value
Return an object of class SNPGDSFileClass.

Author(s)
Xiuwen Zheng

See Also
snpdgsClose
snpgdsOption

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

genofile

# close the file
snpgdsClose(genofile)

---

snpgdsOption  Option settings: chromosome coding, etc

Description

Return an option list used by the SNPRelate package or a GDS file

Usage

snpgdsOption(gdsobj=NULL, autosome.start=1L, autosome.end=22L, ...)

Arguments

gdsobj  an object of class `SNPGDSFileClass`, a SNP GDS file
autosome.start  the starting index of autosome
autosome.end  the ending index of autosome
...  optional arguments for new chromosome coding

Value

A list

Author(s)

Xiuwen Zheng

Examples

# define the new chromosomes 'Z' and 'W'
snpgdsOption(Z=27L, W=28L)

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

snpgdsOption(genofile)

# close the genotype file
snpgdsClose(genofile)
Calculate Identity-By-Descent (IBD) Coefficients

Description

Calculate the three IBD coefficients ($k_0$, $k_1$, $k_2$) for non-inbred individual pairs by Maximum Likelihood Estimation (MLE) or PLINK Method of Moment (MoM).

Usage

```r
snpgdsPairIBD(geno1, geno2, allele.freq, method=c("EM", "downhill.simplex", "MoM"), kinship.constraint=FALSE, max.niter=1000, reltol=sqrt(.Machine$double.eps), coeff.correct=TRUE, out.num.iter=TRUE, verbose=TRUE)
```

Arguments

- `geno1` the SNP genotypes for the first individual, 0 – BB, 1 – AB, 2 – AA, other values – missing
- `geno2` the SNP genotypes for the second individual, 0 – BB, 1 – AB, 2 – AA, other values – missing
- `allele.freq` the allele frequencies
- `method` "EM", "downhill.simplex", or "MoM", see details
- `kinship.constraint` if TRUE, constrict IBD coefficients ($k_0,k_1,k_2$) in the genealogical region ($2 k_0 k_1 >= k_2^2$)
- `max.niter` the maximum number of iterations
- `reltol` relative convergence tolerance; the algorithm stops if it is unable to reduce the value of log likelihood by a factor of $reltol * (abs(log likelihood with the initial parameters) + reltol)$ at a step.
- `coeff.correct` TRUE by default, see details
- `out.num.iter` if TRUE, output the numbers of iterations
- `verbose` if TRUE, show information

Details

If `method = "MoM"`, then PLINK Method of Moment without a allele-count-based correction factor is conducted. Otherwise, two numeric approaches for maximum likelihood estimation can be used: one is Expectation-Maximization (EM) algorithm, and the other is Nelder-Mead method or downhill simplex method. Generally, EM algorithm is more robust than downhill simplex method.

If `coeff.correct` is TRUE, the final point that is found by searching algorithm (EM or downhill simplex) is used to compare the six points (fullsib, offspring, halfsib, cousin, unrelated), since any numeric approach might not reach the maximum position after a finit number of steps. If any of these six points has a higher value of log likelihood, the final point will be replaced by the best one.
Value

Return a data.frame:

k0  IBD coefficient, the probability of sharing ZERO IBD
k1  IBD coefficient, the probability of sharing ONE IBD
loglik  the value of log likelihood
niter  the number of iterations

Author(s)

Xiuwen Zheng

References

Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, Maller J, Sklar P, de Bakker PIW, Daly MJ & Sham PC. 2007. PLINK: a toolset for whole-genome association and population-based linkage analysis. American Journal of Human Genetics, 81.

See Also

snpGdsPairIBDMLELogLik, snpgdsIBDMLE, snpgdsIBDMLELogLik, snpgdsIBDmom

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpGdsExampleFileName())
YRI.id <- read.gdsn(index.gdsn(genofile, "sample.id"))[read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))=='YRI']

# SNP pruning
set.seed(10)
snpset <- snpgdsLDpruning(genofile, sample.id=YRI.id, maf=0.05, missing.rate=0.05)
snpset <- unname(sample(unlist(snpset), 250))

# the number of samples
n <- 25

# specify allele frequencies
RF <- snpgdsSNPRateFreq(genofile, sample.id=YRI.id, snp.id=snpset, with.id=TRUE)
summary(RF$AlleleFreq)

subMLE <- snpgdsIBDMLE(genofile, sample.id=YRI.id[1:n], snp.id=RF$snp.id, allele.freq=RF$AlleleFreq)
subMoM <- snpgdsIBDmom(genofile, sample.id=YRI.id[1:n], snp.id=RF$snp.id, allele.freq=RF$AlleleFreq)
# genotype matrix
mat <- snpgdsGetGeno(genofile, sample.id=YRI.id[1:n], snp.id=snpset, snpfirstdim=TRUE)

########################################################################
rv <- NULL
for (i in 2:n)
{
  rv <- rbind(rv, snpgdsPairIBD(mat[,1], mat[,i], RF$AlleleFreq, "EM")
  print(snpdfsPairIBDMALELogLik(mat[,1], mat[,i], RF$AlleleFreq, relatedness="unrelated", verbose=TRUE))
}
rv
summary(rv$k0 - subMLE$k0[1, 2:n])
summary(rv$k1 - subMLE$k1[1, 2:n])
# ZERO
rv <- NULL
for (i in 2:n)
  rv <- rbind(rv, snpgdsPairIBD(mat[,1], mat[,i], RF$AlleleFreq, "MoM")
rv
summary(rv$k0 - subMoM$k0[1, 2:n])
summary(rv$k1 - subMoM$k1[1, 2:n])
# ZERO

# close the genotype file
snpgdsClose(genofile)

---

**snpdfsPairIBDMALELogLik**

Log likelihood for MLE method in the Identity-By-Descent (IBD) Analysis

**Description**

Calculate the log likelihood values from maximum likelihood estimation.

**Usage**

snpdfsPairIBDMALELogLik(geno1, geno2, allele.freq, k0=NaN, k1=NaN, relatedness=c("","self","fullsib","offspring","halfsib","cousin","unrelated"), verbose=TRUE)

**Arguments**

- **geno1**: the SNP genotypes for the first individual, 0 – BB, 1 – AB, 2 – AA, other values – missing
- **geno2**: the SNP genotypes for the second individual, 0 – BB, 1 – AB, 2 – AA, other values – missing
**snpgdsPairIBDMLELogLik**

allele.freq the allele frequencies
k0 specified IBD coefficient
k1 specified IBD coefficient
relatedness specify a relatedness, otherwise use the values of k0 and k1
verbose if TRUE, show information

**Details**

If (relatedness == "") and (k0 == NaN or k1 == NaN), then return the log likelihood values for each (k0, k1) stored in ibdobj.
If (relatedness == "") and (k0 != NaN) and (k1 != NaN), then return the log likelihood values for a specific IBD coefficient (k0, k1).
If relatedness is: "self", then k0 = 0, k1 = 0; "fullsib", then k0 = 0.25, k1 = 0.5; "offspring", then k0 = 0, k1 = 1; "halfsib", then k0 = 0.5, k1 = 0.5; "cousin", then k0 = 0.75, k1 = 0.25; "unrelated", then k0 = 1, k1 = 0.

**Value**

The value of log likelihood.

**Author(s)**

Xiuwen Zheng

**References**


**See Also**

snpgdsPairIBD, snpgdsIBDMLE, snpgdsIBDMLELogLik, snpgdsIBDMoM

**Examples**

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

YRI.id <- read.gdsn(index.gdsn(genofile, "sample.id"))[
  read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))=='YRI']

# SNP pruning
set.seed(10)
snpset <- snpgdsLDpruning(genofile, sample.id=YRI.id, maf=0.05,
                         missing.rate=0.05)
snpset <- unname(sample(unlist(snpset), 250))

# the number of samples
n <- 25
```
# specify allele frequencies
RF <- snpgdsSNPRateFreq(genofile, sample.id=YRI.id, snp.id=snpset, with.id=TRUE)
snpgdsPairScore

summary(RF$AlleleFreq)

subMLE <- snpgdsIBDMLE(genofile, sample.id=YRI.id[1:n], snp.id=RF$snp.id, allele.freq=RF$AlleleFreq)
subMoM <- snpgdsIBDMoM(genofile, sample.id=YRI.id[1:n], snp.id=RF$snp.id, allele.freq=RF$AlleleFreq)

# genotype matrix
mat <- snpgdsGetGeno(genofile, sample.id=YRI.id[1:n], snp.id=snpset, snpfirstdim=TRUE)

########################################################################

rv <- NULL
for (i in 2:n)
{
  rv <- rbind(rv, snpgdsPairIBD(mat[,1], mat[,i], RF$AlleleFreq, "EM"))
  print(snpdgdsPairIBDMLELogLik(mat[,1], mat[,i], RF$AlleleFreq, relatedness="unrelated", verbose=TRUE))
}

rv
summary(rv$k0 - subMLE$k0[1, 2:n])
summary(rv$k1 - subMLE$k1[1, 2:n])

# ZERO
rv <- NULL
for (i in 2:n)
  rv <- rbind(rv, snpgdsPairIBD(mat[,1], mat[,i], RF$AlleleFreq, "MoM"))

rv
summary(rv$k0 - subMoM$k0[1, 2:n])
summary(rv$k1 - subMoM$k1[1, 2:n])

# close the genotype file
snpgdsClose(genofile)

---

**snpdgdsPairScore**  
**Genotype Score for Pairs of Individuals**

**Description**

Calculate the genotype score for pairs of individuals based on identity-by-state (IBS) measure

**Usage**

snpdgdsPairScore(gdsobj, sample1.id, sample2.id, snp.id=NULL, method=c("IBS", "GVH", "HVG"),
  type=c("per.pair", "per.snp", "matrix", "gds.file"),
  dosage=TRUE, with.id=TRUE, output=NULL, verbose=TRUE)
Arguments

gdsobj a vector of sample id specifying selected samples; if NULL, all samples are used
sample1.id a vector of sample id specifying selected SNPs; if NULL, all SNPs are used
sample2.id a vector of sample id specifying selected SNPs; if NULL, all SNPs are used
snp.id a vector of sample id specifying selected samples; if NULL, all samples are used
method "IBS" – identity-by-state score, "GVH" or "HVG", see Details
type "per.pair", "per.snp" or "matrix", see Value
dosage TRUE, uses dosages 0, 1, 2; FALSE, uses 0, 1 (changing a return value of 1 or 2 to be 1)
with.id if TRUE, returns "sample.id" and "snp.id"; see Value
output if type="gds.file", the file name
verbose if TRUE, show information

Details

<table>
<thead>
<tr>
<th>Patient (sample1.id)</th>
<th>Coded Genotype</th>
<th>Donor (sample2.id)</th>
<th>Coded Genotype</th>
<th>IBS</th>
<th>GVH</th>
<th>HVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>0</td>
<td>AA</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AA</td>
<td>0</td>
<td>AB</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>AA</td>
<td>0</td>
<td>BB</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>AB</td>
<td>1</td>
<td>AA</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AB</td>
<td>1</td>
<td>AB</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AB</td>
<td>1</td>
<td>BB</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BB</td>
<td>2</td>
<td>AA</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BB</td>
<td>2</td>
<td>AB</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BB</td>
<td>2</td>
<td>BB</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Value

Return a list:

sample.id the sample ids used in the analysis, if with.id=TRUE
snp.id the SNP ids used in the analysis, if with.id=TRUE
score a matrix of genotype score: if type="per.pair", a data.frame with the first column for average scores, the second column for standard deviation and the third column for the valid number of SNPs; the additional columns for pairs of samples. if type="per.snp", a 3-by-# of SNPs matrix with the first row for average scores, the second row for standard deviation and the third row for the valid number of individual pairs; if type="matrix", a # of pairs-by-# of SNPs matrix with rows for pairs of individuals

Author(s)

Xiuwen Zheng
References


See Also

snpgdsIBS

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

# autosomal SNPs
selsnp <- snpgdsSelectSNP(genofile, autosome.only=TRUE, remove.monosnp=FALSE)

# sample ID
sample.id <- read.gdsn(index.gdsn(genofile, "sample.id"))
father.id <- read.gdsn(index.gdsn(genofile, "sample.annot/father.id"))

offspring.id <- sample.id[father.id != ""]
father.id <- father.id[father.id != ""]

# calculate average genotype scores
z1 <- snpgdsPairScore(genofile, offspring.id, father.id, snp.id=selsnp, method="IBS", type="per.pair")
names(z1)
head(z1$score)

# calculate average genotype scores
z1 <- snpgdsPairScore(genofile, offspring.id, father.id, snp.id=selsnp, method="IBS", type="per.pair", dosage=FALSE)
names(z1)
head(z1$score)

# calculate average genotype scores
z2 <- snpgdsPairScore(genofile, offspring.id, father.id, snp.id=selsnp, method="IBS", type="per.snp")
names(z2)
mean(z2$score["Avg",])
mean(z2$score["SD",])

plot(z2$score["Avg",], pch=20, cex=0.75, xlab="SNP Index", ylab="IBS score")

# calculate a matrix of genotype scores over samples and SNPs
z3 <- snpgdsPairScore(genofile, offspring.id, father.id, snp.id=selsnp, method="IBS", type="matrix")
dim(z3$score)

# output the score matrix to a GDS file
snpgdsPairScore(genofile, offspring.id, father.id, snp.id=selsnp,
snpgdsPCA

Principal Component Analysis (PCA) on SNP genotype data

Description
To calculate the eigenvectors and eigenvalues for principal component analysis in GWAS.

Usage

```r
snpgdsPCA(gdsobj, sample.id=NULL, snp.id=NULL,
          autosome.only=TRUE, remove.monosnp=TRUE, maf=NaN, missing.rate=NaN,
          algorithm=c("exact", "randomized"),
          eigen.cnt=ifelse(identical(algorithm, "randomized"), 16L, 32L),
          num.thread=1L, bayesian=FALSE, need.genmat=FALSE,
          genmat.only=FALSE, eigen.method=c("DSPEVX", "DSPEV"),
          aux.dim=eigen.cnt*2L, iter.num=10L, verbose=TRUE)
```

Arguments

- `gdsobj`: an object of class `SNPGDSFileClass`, a SNP GDS file
- `sample.id`: a vector of sample id specifying selected samples; if NULL, all samples are used
- `snp.id`: a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
- `autosome.only`: if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
- `remove.monosnp`: if TRUE, remove monomorphic SNPs
- `maf`: to use the SNPs with ">= maf" only; if NaN, no MAF threshold
- `missing.rate`: to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
- `eigen.cnt`: to output the number of eigenvectors; if eigen.cnt <= 0, then return all eigenvectors
- `algorithm`: "exact", traditional exact calculation; "randomized", fast PCA with randomized algorithm introduced in Galinsky et al. 2016
- `num.thread`: the number of (CPU) cores used; if NA, detect the number of cores automatically
- `bayesian`: if TRUE, use bayesian normalization
- `need.genmat`: if TRUE, return the genetic covariance matrix
- `genmat.only`: return the genetic covariance matrix only, do not compute the eigenvalues and eigenvectors
- `eigen.method`: "DSPEVX", "DSPEV"
- `aux.dim`: eigen.cnt*2L
- `iter.num`: 10L
- `verbose`: TRUE
eigen.method  "DSPEVX" – compute the top `eigen.cnt` eigenvalues and eigenvectors using LAPACK::DSPEVX; "DSPEV" – to be compatible with SNPRelate_1.1.6 or earlier, using LAPACK::DSPEV; "DSPEVX" is significantly faster than "DSPEV" if only top principal components are of interest

aux.dim auxiliary dimension used in fast randomized algorithm
iter.num iteration number used in fast randomized algorithm
verbose if TRUE, show information
x a `snpgdsPCAClass` object
eig indices of eigenvectors, like 1:2 or 1:4
... the arguments passed to or from other methods, like `pch`, `col`

Details

The minor allele frequency and missing rate for each SNP passed in `snp.id` are calculated over all the samples in `sample.id`.

Value

Return a `snpgdsPCAClass` object, and it is a list:

- `sample.id` the sample ids used in the analysis
- `snp.id` the SNP ids used in the analysis
- `eigenval` eigenvalues
- `eigenvect` eigenvectors, "# of samples" x "eigen.cnt"
- `varprop` variance proportion for each principal component
- `TraceXTX` the trace of the genetic covariance matrix
- `Bayesian` whether use bayesian normalization
- `genmat` the genetic covariance matrix

Author(s)

Xiuwen Zheng

References


See Also

`snpgdsPCACorr`, `snpgdsPCASampLoading`, `snpgdsPCASNPLoading`
Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

# run PCA
RV <- snpgdsPCA(genofile)

# eigenvalues
head(RV$eigenval)

# variance proportion (%)
head(round(RV$varprop*100, 2))
# [1] 12.23 5.84 1.01 0.95 0.84 0.74

plot(RV)
plot(RV, 1:4)

#### there is no population information ####

# make a data.frame
tab <- data.frame(sample.id = RV$sample.id,
                  EV1 = RV$eigenvect[,1], # the first eigenvector
                  EV2 = RV$eigenvect[,2], # the second eigenvector
                  stringsAsFactors = FALSE)

head(tab)
# sample.id  EV1    EV2
# 1  NA19152 -0.08411287 -0.01226860
# 2  NA19139 -0.08360644 -0.01085849
# 3  NA18912 -0.08110808 -0.01184524
# 4  NA19160 -0.08688064 -0.01447106
# 5  NA07034  0.03109761  0.07709255
# 6  NA07055  0.03228450  0.08155730

# draw
plot(tab$EV2, tab$EV1, xlab="eigenvector 2", ylab="eigenvector 1")

#### there are population information ####

# get population information
# or pop_code <- scan("pop.txt", what=character())
# if it is stored in a text file "pop.txt"
pop_code <- read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))

# get sample id
samp.id <- read.gdsn(index.gdsn(genofile, "sample.id"))

# assume the order of sample IDs is as the same as population codes
cbind(samp.id, pop_code)
# samp.id  pop_code
# [1,] "NA19152"  "YRI"
# [2,] "NA19139"  "YRI"
# [3,] "NA18912"  "YRI"
snpgdsPCACorr

PC-correlated SNPs in principal component analysis

Description

To calculate the SNP correlations between eigenvectors and SNP genotypes

Usage

snpgdsPCACorr(pcaobj, gdsobj, snp.id=NULL, eig.which=NULL, num.thread=1L, verbose=TRUE)

Arguments

pcaobj the snpgdsPCAClass object returned from the function snpgdsPCA
gdsobj an object of class SNPGDSFileClass, a SNP GDS file
snp.id a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
eig.which a vector of integers, to specify which eigenvectors to be used
num.thread the number of (CPU) cores used; if NA, detect the number of cores automatically
verbose if TRUE, show information
snpgdsPCASampLoading

Description

To calculate the sample eigenvectors using the specified SNP loadings

Usage

```r
snpgdsPCASampLoading(loadobj, gdsobj, sample.id=NULL, num.thread=1L, verbose=TRUE)
```

Arguments

- `loadobj`: the `snpgdsPCASNPLoadingClass` object, returned from `snpgdsPCASNPLoading`
- `gdsobj`: an object of class `SNPGDSFileClass`, a SNP GDS file
- `sample.id`: a vector of sample id specifying selected samples; if NULL, all samples are used
- `num.thread`: the number of CPU cores used
- `verbose`: if TRUE, show information

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())
# get chromosome index
chr <- read.gdsn(index.gdsn(genofile, "snp.chromosome"))

pca <- snpgdsPCA(genofile)
CORR <- snpgdsPCACorr(pca, genofile, eig.which=1:4)
plot(abs(CORR$snpcorr[3,]), xlab="SNP Index", ylab="PC 3", col=chr)

# close the file
snpgdsClose(genofile)
```
Details

The sample.id are usually different from the samples used in the calculation of SNP loadings.

Value

Return a snpgdsPCAClass object, and it is a list:

- `sample.id` the sample ids used in the analysis
- `snp.id` the SNP ids used in the analysis
- `eigenval` eigenvalues
- `eigenvect` eigenvectors, "# of samples" x "eigen.cnt"
- `TraceXTX` the trace of the genetic covariance matrix
- `Bayesian` whether use bayerisan normalization

Author(s)

Xiuwen Zheng

References


See Also

`snpgdsPCA`, `snpgdsPCACorr`, `snpgdsPCASNPLoading`

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgcdsExampleFileName())
sample.id <- read.gdsn(index.gdsn(genofile, "sample.id"))
PCARV <- snpgdsPCA(genofile, eigen.cnt=8)
SnpLoad <- snpgdsPCASNPLoading(PCARV, genofile)

# calculate sample eigenvectors from SNP loadings
SL <- snpgdsPCASampLoading(SnpLoad, genofile, sample.id=sample.id[1:100])
diff <- PCARV$eigenvect[1:100,] - SL$eigenvect
summary(c(diff))
# ~ ZERO

# close the genotype file
snpgdsClose(genofile)
```
Description

To calculate the SNP loadings in Principal Component Analysis

Usage

```r
snpgdsPCASNPLoading(pcaobj, gdsobj, num.thread=1L, verbose=TRUE)
```

Arguments

- `pcaobj`: the `snpgdsPCAClass` object returned from the function `snpgdsPCA`.
- `gdsobj`: an object of class `SNPGDSFileClass`, a SNP GDS file.
- `num.thread`: the number of (CPU) cores used; if `NA`, detect the number of cores automatically.
- `verbose`: if `TRUE`, show information.

Details

Calculate the SNP loadings (or SNP eigenvectors) from the principal component analysis conducted in `snpgdsPCA`.

Value

Return a `snpgdsPCASNPLoading` object, which is a list:

- `sample.id`: the sample ids used in the analysis.
- `snp.id`: the SNP ids used in the analysis.
- `eigenval`: eigenvalues.
- `snploading`: the SNP loadings, or SNP eigenvectors.
- `TraceXTX`: the trace of the genetic covariance matrix.
- `Bayesian`: whether use bayesian normalization.
- `avefreq`: the allele frequency used in `snpgdsPCA`.
- `scale`: internal parameter.

Author(s)

- Xiuwen Zheng

References

Conversion from PLINK PED to GDS

Convert a PLINK PED text file to a GDS file.

Usage

```
snpdgsPED2GDS(ped.fn, map.fn, out.gdsfn, family=TRUE, snpfirstdim=FALSE, compress.annotation="ZIP_RA.max", compress.geno="", verbose=TRUE)
```

Arguments

- `ped.fn`: the file name of PED file, genotype information
- `map.fn`: the file name of MAP file
- `out.gdsfn`: the output GDS file
- `family`: if `TRUE`, to include family information in the sample annotation
- `snpfirstdim`: if `TRUE`, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc)
- `compress.annotation`: the compression method for the GDS variables, except "genotype"; optional values are defined in the function `add.gdsn`
- `compress.geno`: the compression method for "genotype"; optional values are defined in the function `add.gdsn`
- `verbose`: if `TRUE`, show information
Details

GDS – Genomic Data Structures, the extended file name used for storing genetic data, and the file format is used in the gdsfmt package.

PED – PLINK PED format.

Value

None.

Author(s)

Xiuwen Zheng

References

Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, Maller J, Sklar P, de Bakker PIW, Daly MJ & Sham PC. 2007. PLINK: a toolset for whole-genome association and population-based linkage analysis. American Journal of Human Genetics, 81.

See Also

snpgdsGDS2PED, snpgdsBED2GDS, snpgdsGDS2BED

Examples

# open
-genofile <- snpgdsOpen(snpExampleFileName())

snpgdsGDS2PED(genofile, "tmp")

# close
snpgdsClose(genofile)

# PED ==> GDS
snpgdsPED2GDS("tmp.ped", "tmp.map", "test.gds")

# delete the temporary file
unlink(c("tmp.ped", "tmp.map", "test.gds"), force=TRUE)

---

**snpgdsSampMissRate**  
**Missing Rate of Samples**

Description

Return the missing fraction for each sample

Usage

snpgdsSampMissRate(gdsobj, sample.id=NULL, snp.id=NULL, with.id=FALSE)
Arguments

gdsobj an object of class \texttt{SNPGDSFileClass}, a SNP GDS file
sample.id a vector of sample id specifying selected samples; if NULL, all samples will be used
snp.id a vector of snp id specifying selected SNPs; if NULL, all SNPs will be used
with.id if TRUE, the returned value with sample id

Value

A vector of numeric values.

Author(s)

Xiuwen Zheng

See Also

\texttt{snpdfsSNPRateFreq}

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdfsExampleFileName())

RV <- snpgdsSampMissRate(genofile)
summary(RV)

# close the genotype file
snpgdsClose(genofile)
Function 

**snpgdsSlidingWindow**

Description

Apply a user-defined function with a sliding window.

Usage

```r
snpgdsSlidingWindow(gdsobj, sample.id=NULL, snp.id=NULL, 
                FUN=NULL, winsize=100000L, shift=10000L, unit=c("basepair", "locus"), 
                winstart=NULL, autosome.only=FALSE, remove.monosnp=TRUE, maf=NaN, 
                missing.rate=NaN, as.is=c("list", "numeric", "array"), 
                with.id=c("snp.id", "snp.id.in.window", "none"), num.thread=1, 
                verbose=TRUE, ...)```

Arguments

- `gdsobj`: Genotype data object.
- `sample.id`, `snp.id`: Optional sample and SNP identifiers.
- `FUN`: User-defined function to apply within the sliding window.
- `winsize`, `shift`: Window size and sliding step, respectively.
- `unit`: Unit of measurement for `winsize` and `shift`.
- `winstart`: Optional starting position for the sliding window.
- `autosome.only`: If TRUE, only autosomal SNPs are considered.
- `remove.monosnp`: If TRUE, remove monomorphic SNPs.
- `maf`: Minimum allele frequency threshold.
- `missing.rate`: Maximum missing rate threshold.
- `as.is`: Type of output, list, numeric, or array.
- `with.id`: If TRUE, include SNP identifiers.
- `num.thread`: Number of threads for parallel processing.
- `verbose`: If TRUE, show processing information.

Value

Return a list of SNP IDs.

Author(s)

Xiuwen Zheng

See Also

- `snpgdsSampMissRate`
- `snpgdsSNPRateFreq`
- `snpgdsLDpruning`

Examples

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

snpset <- snpgdsSelectSNP(genofile, maf=0.05, missing.rate=0.95)
length(snpset)
# 7502

# close the genotype file
snpgdsClose(genofile)
```
Arguments

gdsobj an object of class `SNPGDSFileClass`, a SNP GDS file
sample.id a vector of sample id specifying selected samples; if NULL, all samples are used
snp.id a vector of snp id specifying selected SNPs; if NULL, all SNPs are used
FUN a character or a user-defined function, see details
winsize the size of sliding window
shift the amount of shifting the sliding window
unit "basepair"—winsize and shift are applied with SNP coordinate of basepair;
"locus"—winsize and shift are applied according to the SNP order in the GDS file
winstart NULL—no specific starting position; an integer—a starting position for all chromosomes; or a vector of integer—the starting positions for each chromosome
autosome.only if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
remove.monosnp if TRUE, remove monomorphic SNPs
maf to use the SNPs with ">= maf" only; if NaN, no MAF threshold
missing.rate to use the SNPs with "<= missing.rate" only; if NaN, no missing threshold
as.is save the value returned from FUN as "list" or "numeric"; "array" is equivalent to "numeric" except some cases, see details
with.id "snp.id", "snp.id.in.window" or "none"
num.thread the number of (CPU) cores used; if NA, detect the number of cores automatically
verbose if TRUE, show information
... optional arguments to FUN

Details

If FUN="snpgdsFst", two additional arguments "population" and "method" should be specified. "population" and "method" are defined in `snpgdsFst`. "as.is" could be "list" (returns a list of the values from `snpgdsFst`), "numeric" (population-average Fst, returns a vector) or "array" (population-average and -specific Fst, returns a '# of pop + 1'-by-'# of windows' matrix, and the first row is population-average Fst).

Value

Return a list

Author(s)

Xiuwen Zheng

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpGdsExampleFileName())

# sliding windows
rv <- snpgdsSlidingWindow(genofile, winsize=500000, shift=100000,
FUN=function(...) NULL)
# Write a script for plotting and printing SNP frequencies

```r
# plot
plot(rv$chr1.num, ylab="# of SNPs in the sliding window")

# close the genotype file
snpgdsClose(genofile)
```

---

**snpdfsSNPList**

*Create a SNP list object*

**Description**

A list object of SNP information including rs, chr, pos, allele and allele frequency.

**Usage**

```r
snpdfsSNPList(gdsobj, sample.id=NULL)
```

**Arguments**

- `gdsobj` an object of class `SNPGDSFileClass`, a SNP GDS file
- `sample.id` a vector of sample id specifying selected samples; if NULL, all samples are used

**Value**

Return an object of `snpdfsSNPListClass` including the following components:

- `rs.id` SNP id
- `chromosome` SNP chromosome index
- `position` SNP physical position in basepair
- `allele` reference / non-ref alleles
- `afreq` allele frequency

**Author(s)**

Xiuwen Zheng

**See Also**

`snpdfsSNPListIntersect, snpgdsSNPListStrand`

**Examples**

```r
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpdfsExampleFileName())

# to get a snp list object
snplist <- snpgdfsSNPList(genofile)

# close the file
snpgdfsClose(genofile)
```
The class of a SNP list, and its instance is returned from `snpgdsSNPList`.

**Value**

Return an object of “`snpgdsSNPListClass`” including the following components:

- `rs.id` SNP id
- `chromosome` SNP chromosome index
- `position` SNP physical position in basepair
- `allele` reference / non-ref alleles
- `afreq` allele frequency

**Author(s)**

Xiuwen Zheng

**See Also**

`snpgdsSNPList`, `snpgdsSNPListIntersect`

Get a common SNP list by comparing their rs id, chromosome indices and positions.

**Usage**

`snpgdsSNPListIntersect(snplist1, snplist2)`

**Arguments**

- `snplist1` the first SNP list object `snpgdsSNPListClass`
- `snplist2` the second SNP list object `snpgdsSNPListClass`

**Value**

Return an object of `snpgdsSNPListClass` including the following components:

- `rs.id` SNP id
- `chromosome` SNP chromosome index
- `position` SNP physical position in basepair
- `allele` reference / non-ref alleles from the first SNP list object
- `afreq` allele frequency from the first SNP list object
snpgdsSNPListStrand

Author(s)
Xiuwen Zheng

See Also
snpgdsSNPList, snpgdsSNPListStrand

Examples
# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

# to get a snp list object
snplist1 <- snpgdsSNPList(genofile)
snplist2 <- snpgdsSNPList(genofile)

# a common snp list
snplist <- snpgdsSNPListIntersect(snplist1, snplist2)

summary(snplist$afreq)

# close the file
snpgdsClose(genofile)

snpgdsSNPListStrand Switch allele strand.

Description
To get a logical vector, indicating whether allele references of snplist2 need to be switched, with respect to snplist1.

Usage
snpgdsSNPListStrand(snplist1, snplist2, same.strand=FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snplist1</td>
<td>the first SNP list object “snpgdsSNPListClass”</td>
</tr>
<tr>
<td>snplist2</td>
<td>the second SNP list object “snpgdsSNPListClass”</td>
</tr>
<tr>
<td>same.strand</td>
<td>TRUE assuming alleles are on the same strand (e.g., forward strand); otherwise, FALSE not assuming whether on the same strand or not</td>
</tr>
</tbody>
</table>

Value
a logical vector, where TRUE indicates the allele references need to be switched, and NA indicates that locus is not in the common snp list.

Author(s)
Xiuwen Zheng
\texttt{snpgdsSNPRateFreq}  

\textbf{Description}  

Calculate the allele frequency, minor allele frequency and missing rate per SNP.

\textbf{Usage}  

\texttt{snpgdsSNPRateFreq(gdsobj, sample.id=NULL, snp.id=NULL, with.id=FALSE)}

\textbf{Arguments}  

- \texttt{gdsobj}  
  an object of class \texttt{SNPGDSFileClass}, a SNP GDS file
- \texttt{sample.id}  
  a vector of sample id specifying selected samples; if NULL, all samples will be used
- \texttt{snp.id}  
  a vector of snp id specifying selected SNPs; if NULL, all SNPs will be used
- \texttt{with.id}  
  if TRUE, return sample and SNP IDs

\textbf{Value}  

Return a list:

- \texttt{AlleleFreq}  
  allele frequencies
- \texttt{MinorFreq}  
  minor allele frequencies
- \texttt{MissingRate}  
  missing rates
- \texttt{sample.id}  
  sample id, if with.id=TRUE
- \texttt{snp.id}  
  SNP id, if with.id=TRUE
snpgdsSummary

Author(s)
Xiuwen Zheng

See Also
snpgdsSampMissRate

Examples

# open an example dataset (HapMap)
genofile <- snpgdsOpen(snpgdsExampleFileName())

RV <- snpgdsSNPRateFreq(genofile)
hist(RV$AlleleFreq, breaks=128)
summary(RV$MissingRate)

# close the file
snpgdsClose(genofile)

snpgdsSummary  Summary of GDS genotype file

Description
Print the information stored in the gds object

Usage
snpgdsSummary(gds, show=TRUE)

Arguments
gds  a GDS file name, or an object of class SNPGDSFileClass
show  if TRUE, show information

Value
Return a list:
sample.id  the IDs of valid samples
snp.id  the IDs of valid SNPs

Author(s)
Xiuwen Zheng

Examples
snpgdsSummary(snpgdsExampleFileName())
**snpgdsTranspose**

*Transpose genotypic matrix*

**Description**

Transpose the genotypic matrix if needed.

**Usage**

`snpgdsTranspose(gds.fn, snpfirstdim=FALSE, compress=NULL, optimize=TRUE, verbose=TRUE)`

**Arguments**

- `gds.fn` the file name of SNP GDS format
- `snpfirstdim` if TRUE, genotypes are stored in snp-by-sample; if FALSE, sample-by-snp mode; if NA, force to transpose the SNP matrix
- `compress` the compression mode for SNP genotypes, optional values are defined in the function of add.gdsn; if NULL, to use the compression mode
- `optimize` if TRUE, call `cleanup.gds` after transposing
- `verbose` if TRUE, show information

**Value**

None.

**Author(s)**

Xiuwen Zheng

**Examples**

```r
# the file name of SNP GDS
(fn <- snpgdsExampleFileName())

# copy the file
dir.copy(fn, "test.gds", overwrite=TRUE)

# summary
snpgdsSummary("test.gds")

# transpose the SNP matrix
snpgdsTranspose("test.gds", snpfirstdim=TRUE)

# summary
snpgdsSummary("test.gds")

# delete the temporary file
dir.copy("test.gds", force=TRUE)
```
snpgdsVCF2GDS

Reformat VCF file(s)

Description
Reformat Variant Call Format (VCF) file(s)

Usage
snpgdsVCF2GDS(vcf.fn, out.fn, method=c("biallelic.only", "copy.num.of.ref"),
method=c("biallelic.only", "copy.num.of.ref"),
snpfirstdim=FALSE, compress.annotation="ZIP_RA.max", compress.geno="",
ref.allele=NULL, ignore.chr.prefix="chr", verbose=TRUE)

Arguments
vcf.fn the file name of VCF format, vcf.fn can be a vector, see details
out.fn the file name of output GDS
method either "biallelic.only" by default or "copy.num.of.ref", see details
snpfirstdim if TRUE, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc)
compress.annotation the compression method for the GDS variables, except "genotype"; optional values are defined in the function add.gdsn
compress.geno the compression method for "genotype"; optional values are defined in the function add.gdsn
ref.allele NULL or a character vector indicating reference allele (like "A", "G", "T", NA, ...) for each site where NA to use the original reference allele in the VCF file(s). The length of character vector should be the total number of variants in the VCF file(s).
ignore.chr.prefix a vector of character, indicating the prefix of chromosome which should be ignored, like "chr"; it is not case-sensitive
verbose if TRUE, show information

Details
GDS – Genomic Data Structures used for storing genetic array-oriented data, and the file format used in the gdsfmt package.
VCF – The Variant Call Format (VCF), which is a generic format for storing DNA polymorphism data such as SNPs, insertions, deletions and structural variants, together with rich annotations.

If there are more than one file names in vcf.fn, snpgdsVCF2GDS will merge all dataset together if they all contain the same samples. It is useful to combine genetic/genomic data together if VCF data are divided by chromosomes.

method = "biallelic.only": to exact bi-allelic and polymorphic SNP data (excluding monomorphic variants); method = "copy.num.of.ref": to extract and store dosage (0, 1, 2) of the reference allele for all variant sites, including bi-allelic SNPs, multi-allelic SNPs, indels and structural variants.
Haploid and triploid calls are allowed in the transfer, the variable `snp.id` stores the original row index of variants, and the variable `snp.rs.id` stores the rs id.

When `snp.chromosome` in the GDS file is character, SNPRelate treats a chromosome as autosome only if it can be converted to a numeric value (like "1", "22"). It uses "X" and "Y" for non-autosomes instead of numeric codes. However, some software format chromosomes in VCF files with a prefix "chr". Users should remove that prefix when importing VCF files by setting `ignore.chr.prefix = "chr"`.

**Value**

Return the file name of GDS format with an absolute path.

**Author(s)**

Xiuwen Zheng

**References**


**See Also**

`snpdgsBED2GDS`

**Examples**

```r
# the VCF file
vcf.fn <- system.file("extdata", "sequence.vcf", package="SNPRelate")
cat(readLines(vcf.fn), sep="\n")

snpdgsVCF2GDS(vcf.fn, "test1.gds", method="biallelic.only")
snpdgsSummary("test1.gds")

snpdgsVCF2GDS(vcf.fn, "test2.gds", method="biallelic.only", snpfirstdim=TRUE)
snpdgsSummary("test2.gds")

snpdgsVCF2GDS(vcf.fn, "test3.gds", method="copy.num.of.ref", snpfirstdim=TRUE)
snpdgsSummary("test3.gds")

snpdgsVCF2GDS(vcf.fn, "test4.gds", method="copy.num.of.ref")
snpdgsSummary("test4.gds")

snpdgsVCF2GDS(vcf.fn, "test5.gds", method="copy.num.of.ref", ref.allele=c("A", "T", "T", "T", "A"))
snpdgsSummary("test5.gds")

# open "test1.gds"
(genofile <- snpdgsOpen("test1.gds"))

read.gdsn(index.gdsn(genofile, "sample.id"))
```
read.gdsn(index.gdsn(genofile, "snp.rs.id"))
read.gdsn(index.gdsn(genofile, "genotype"))

# close the file
snpgdsClose(genofile)

# open "test2.gds"
(genofile <- snpgdsOpen("test2.gds"))

read.gdsn(index.gdsn(genofile, "sample.id"))
read.gdsn(index.gdsn(genofile, "snp.rs.id"))
read.gdsn(index.gdsn(genofile, "genotype"))

# close the file
snpgdsClose(genofile)

# open "test3.gds"
(genofile <- snpgdsOpen("test3.gds"))

read.gdsn(index.gdsn(genofile, "sample.id"))
read.gdsn(index.gdsn(genofile, "snp.rs.id"))
read.gdsn(index.gdsn(genofile, "genotype"))

# close the file
snpgdsClose(genofile)

# open "test4.gds"
(genofile <- snpgdsOpen("test4.gds"))

read.gdsn(index.gdsn(genofile, "sample.id"))
read.gdsn(index.gdsn(genofile, "snp.rs.id"))
read.gdsn(index.gdsn(genofile, "snp.allele"))
read.gdsn(index.gdsn(genofile, "genotype"))

# close the file
snpgdsClose(genofile)

# open "test5.gds"
(genofile <- snpgdsOpen("test5.gds"))

read.gdsn(index.gdsn(genofile, "sample.id"))
read.gdsn(index.gdsn(genofile, "snp.rs.id"))
read.gdsn(index.gdsn(genofile, "snp.allele"))
read.gdsn(index.gdsn(genofile, "genotype"))

# close the file
snpgdsClose(genofile)

# delete the temporary files
unlink(paste("test", 1:5, ".gds", sep=""), force=TRUE)
snpgdsVCF2GDS_R
Reformat a VCF file (R implementation)

Description
Reformat a Variant Call Format (VCF) file

Usage

```r
snpgdsVCF2GDS_R(vcf.fn, out.fn, nblock=1024,
  method = c("biallelic.only", "copy.num.of.ref"),
  compress.annotation="ZIP_RA.max", snpfirstdim=FALSE, option = NULL,
  verbose=TRUE)
```

Arguments

- `vcf.fn`: the file name of VCF format, `vcf.fn` can be a vector, see details
- `out.fn`: the output gds file
- `nblock`: the buffer lines
- `method`: either "biallelic.only" by default or "copy.num.of.ref", see details
- `compress.annotation`: the compression method for the GDS variables, except "genotype"; optional values are defined in the function add.gdsn
- `snpfirstdim`: if TRUE, genotypes are stored in the individual-major mode, (i.e, list all SNPs for the first individual, and then list all SNPs for the second individual, etc)
- `option`: NULL or an object from `snpgdsOption`, see details
- `verbose`: if TRUE, show information

Details
GDS – Genomic Data Structures used for storing genetic array-oriented data, and the file format used in the gdsfmt package.
VCF – The Variant Call Format (VCF), which is a generic format for storing DNA polymorphism data such as SNPs, insertions, deletions and structural variants, together with rich annotations.

If there are more than one file name in vcf.fn, snpgdsVCF2GDS will merge all dataset together once they all contain the same samples. It is useful to combine genetic data if VCF data are divided by chromosomes.

- `method = "biallelic.only"`: to exact bi-allelic and polymorphic SNP data (excluding monomorphic variants);
- `method = "copy.num.of.ref"`: to extract and store dosage (0, 1, 2) of the reference allele for all variant sites, including bi-allelic SNPs, multi-allelic SNPs, indels and structural variants.

Haploid and triploid calls are allowed in the transfer, the variable `snp.id` stores the original the row index of variants, and the variable `snp.rs.id` stores the rs id.

The user could use option to specify the range of code for autosomes. For humans there are 22 autosomes (from 1 to 22), but dogs have 38 autosomes. Note that the default settings are used for humans. The user could call option = `snpgdsOption(autosome.end=38)` for importing the VCF file of dog. It also allows defining new chromosome coding, e.g., option = `snpgdsOption(Z=27)`, then "Z" will be replaced by the number 27.
**Value**

None.

**Author(s)**

Xiuwen Zheng

**References**


**See Also**

`snpgdsVCF2GDS_R`, `snpgdsOption`, `snpgdsBED2GDS`

**Examples**

```r
# The VCF file
cat(readLines(vcf.fn), sep="\n")

snpgdsVCF2GDS_R(vcf.fn, "test1.gds", method="biallelic.only")
snpgdsSummary("test1.gds")

snpgdsVCF2GDS_R(vcf.fn, "test2.gds", method="biallelic.only")
snpgdsSummary("test2.gds")

snpgdsVCF2GDS_R(vcf.fn, "test3.gds", method="copy.num.of.ref")
snpgdsSummary("test3.gds")

snpgdsVCF2GDS_R(vcf.fn, "test4.gds", method="copy.num.of.ref")
snpgdsSummary("test4.gds")
```
Index

*Topic GDS
snpgdsAdmixProp, 6
snpgdsAllleleSwitch, 8
snpgdsApartSelection, 9
snpgdsBED2GDS, 10
snpgdsClose, 12
snpgdsCombineGeno, 13
snpgdsCreateGeno, 14
snpgdsCreateGenoSet, 15
snpgdsCutTree, 17
snpgdsDiss, 19
snpgdsDrawTree, 21
snpgdsEIGMIX, 22
snpgdsErrMsg, 24
snpgdsExampleFileName, 25
SNPGDSFileClass, 26
snpgdsFst, 26
snpgdsGDS2BED, 28
snpgdsGDS2Eigen, 29
snpgdsGDS2PED, 30
snpgdsGEN2GDS, 31
snpgdsGetGeno, 33
snpgdsGRM, 34
snpgdsHCluster, 35
snpgdsHWE, 37
snpgdsIBOKING, 38
snpgdsIBDMLE, 40
snpgdsIBDMLLELogLik, 43
snpgdsIBDMinM, 45
snpgdsIBDSelection, 47
snpgdsIBS, 49
snpgdsIBSNum, 50
snpgdsIndInb, 51
snpgdsIndInbCoef, 53
snpgdsIndivBeta, 54
snpgdsLDMat, 55
snpgdsLDpair, 57
snpgdsLDpruning, 58
snpgdsOpen, 60
snpgdsOption, 61
snpgdsPairIBD, 62
snpgdsPairIBDMLLELogLik, 64
snpgdsPairScore, 66
snpgdsPCA, 69
snpgdsPCACorr, 72
snpgdsPCASampLoading, 73
snpgdsPCASNPLoading, 75
snpgdsPED2GDS, 76
snpgdsSampMissRate, 77
snpgdsSelectSNP, 78
snpgdsSlidingWindow, 79
snpgdsSNPList, 81
snpgdsSNPListClass, 82
snpgdsSNPListIntersect, 82
snpgdsSNPListStrand, 83
snpgdsSNPRateFreq, 84
snpgdsSummary, 85
snpgdsTranspose, 86
snpgdsVCF2GDS, 87
SNPRelate-package, 3

*Topic GWAS
snpgdsAdmixProp, 6
snpgdsAllleleSwitch, 8
snpgdsApartSelection, 9
snpgdsBED2GDS, 10
snpgdsClose, 12
snpgdsCombineGeno, 13
snpgdsCreateGeno, 14
snpgdsCreateGenoSet, 15
snpgdsCutTree, 17
snpgdsDiss, 19
snpgdsDrawTree, 21
snpgdsEIGMIX, 22
snpgdsErrMsg, 24
snpgdsExampleFileName, 25
SNPGDSFileClass, 26
snpgdsFst, 26
snpgdsGDS2BED, 28
snpgdsGDS2Eigen, 29
snpgdsGDS2PED, 30
snpgdsGEN2GDS, 31
snpgdsGetGeno, 33
snpgdsGRM, 34
snpgdsHCluster, 35
snpgdsHWE, 37
snpgdsIBOKING, 38
snpgdsIBDMLE, 40
snpgdsIBDMLLELogLik, 43
snpgdsIBDMinM, 45
snpgdsIBDSelection, 47
snpgdsIBS, 49
snpgdsIBSNum, 50
snpgdsIndInb, 51
snpgdsIndInbCoef, 53
snpgdsIndivBeta, 54
snpgdsLDMat, 55
snpgdsLDpair, 57
snpgdsLDpruning, 58
snpgdsOpen, 60
snpgdsOption, 61
snpgdsPairIBD, 62
snpgdsPairIBDMLLELogLik, 64
snpgdsPairScore, 66
snpgdsPCA, 69
snpgdsPCACorr, 72
snpgdsPCASampLoading, 73
snpgdsPCASNPLoading, 75
snpgdsPED2GDS, 76
snpgdsSampMissRate, 77
snpgdsSelectSNP, 78
snpgdsSlidingWindow, 79
snpgdsSNPList, 81
snpgdsSNPListClass, 82
snpgdsSNPListIntersect, 82
snpgdsSNPListStrand, 83
snpgdsSNPRateFreq, 84
snpgdsSummary, 85
snpgdsTranspose, 86
snpgdsVCF2GDS, 87
SNPRelate-package, 3
<table>
<thead>
<tr>
<th>Index Item</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>snpgdsIBDMLE</td>
<td>40</td>
</tr>
<tr>
<td>snpgdsIBDMLELogLik</td>
<td>43</td>
</tr>
<tr>
<td>snpgdsIBDMom</td>
<td>45</td>
</tr>
<tr>
<td>snpgdsIBDSelection</td>
<td>47</td>
</tr>
<tr>
<td>snpgdsIBS</td>
<td>49</td>
</tr>
<tr>
<td>snpgdsIBSNum</td>
<td>50</td>
</tr>
<tr>
<td>snpgdsIndInb</td>
<td>51</td>
</tr>
<tr>
<td>snpgdsIndInbCoef</td>
<td>53</td>
</tr>
<tr>
<td>snpgdsIndivBeta</td>
<td>54</td>
</tr>
<tr>
<td>snpgdsLDMat</td>
<td>55</td>
</tr>
<tr>
<td>snpgdsLDpair</td>
<td>57</td>
</tr>
<tr>
<td>snpgdsLDpruning</td>
<td>58</td>
</tr>
<tr>
<td>snpgdsOpen</td>
<td>60</td>
</tr>
<tr>
<td>snpgdsOption</td>
<td>61</td>
</tr>
<tr>
<td>snpgdsPairIBD</td>
<td>62</td>
</tr>
<tr>
<td>snpgdsPairIBDMLELogLik</td>
<td>64</td>
</tr>
<tr>
<td>snpgdsPairScore</td>
<td>66</td>
</tr>
<tr>
<td>snpgdsPCA</td>
<td>69</td>
</tr>
<tr>
<td>snpgdsPCACorr</td>
<td>72</td>
</tr>
<tr>
<td>snpgdsPCASampLoading</td>
<td>73</td>
</tr>
<tr>
<td>snpgdsPCASNPLoading</td>
<td>75</td>
</tr>
<tr>
<td>snpgdsPED2GDS</td>
<td>76</td>
</tr>
<tr>
<td>snpgdsSampMissRate</td>
<td>77</td>
</tr>
<tr>
<td>snpgdsSelectSNP</td>
<td>78</td>
</tr>
<tr>
<td>snpgdsSlidingWindow</td>
<td>79</td>
</tr>
<tr>
<td>snpgdsSNPList</td>
<td>81</td>
</tr>
<tr>
<td>snpgdsSNPListClass</td>
<td>82</td>
</tr>
<tr>
<td>snpgdsSNPListIntersect</td>
<td>82</td>
</tr>
<tr>
<td>snpgdsSNPListStrand</td>
<td>83</td>
</tr>
<tr>
<td>snpgdsSNPRateFreq</td>
<td>84</td>
</tr>
<tr>
<td>snpgdsSummary</td>
<td>85</td>
</tr>
<tr>
<td>snpgdsTranspose</td>
<td>86</td>
</tr>
<tr>
<td>snpgdsVCF2GDS</td>
<td>87</td>
</tr>
<tr>
<td>snpgdsVCF2GDS_R</td>
<td>90</td>
</tr>
<tr>
<td>SNPGDSFileClass</td>
<td>93</td>
</tr>
</tbody>
</table>

* **Topic** **PCA**
  - snpgdsPCA | 69 |
  - snpgdsPCACorr | 72 |
  - snpgdsPCASampLoading | 73 |
  - snpgdsPCASNPLoading | 75 |

* **Topic** **datasets**
  - hapmap_geno | 5 |

* **Topic** **gds**
  - snpgdsVCF2GDS_R | 90 |

<table>
<thead>
<tr>
<th>Library</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>gds</td>
<td>26, 30</td>
</tr>
<tr>
<td>gdsfmt</td>
<td>11, 28, 29, 31, 32, 77, 87, 90</td>
</tr>
<tr>
<td>hapmap_geno</td>
<td>5</td>
</tr>
<tr>
<td>hclust</td>
<td>36</td>
</tr>
<tr>
<td>plot.snpgdsPCAClass (snpgdsPCA)</td>
<td>69</td>
</tr>
<tr>
<td>snpgdsAdmixProp</td>
<td>6</td>
</tr>
<tr>
<td>snpgdsAlleleSwitch</td>
<td>8</td>
</tr>
<tr>
<td>snpgdsApartSelection</td>
<td>9</td>
</tr>
<tr>
<td>snpgdsBED2GDS</td>
<td>10, 28, 32, 77, 88, 91</td>
</tr>
<tr>
<td>snpgdsClose</td>
<td>12, 26, 60</td>
</tr>
<tr>
<td>snpgdsCombineGeno</td>
<td>13, 15, 16</td>
</tr>
<tr>
<td>snpgdsCreateGeno</td>
<td>13, 14, 16</td>
</tr>
<tr>
<td>snpgdsCreateGenoSet</td>
<td>13, 15, 15</td>
</tr>
<tr>
<td>snpgdsCutTree</td>
<td>17, 21, 22, 36</td>
</tr>
<tr>
<td>snpgdsDiss</td>
<td>18, 19, 36</td>
</tr>
<tr>
<td>snpgdsDrawTree</td>
<td>18, 21</td>
</tr>
<tr>
<td>snpgdsEIGMIX</td>
<td>6, 22, 23</td>
</tr>
<tr>
<td>snpgdsErrMsg</td>
<td>24</td>
</tr>
<tr>
<td>snpgdsExampleFileName</td>
<td>25</td>
</tr>
<tr>
<td>SNPGDSFileClass</td>
<td>8, 12, 19, 23, 26, 28, 29, 33, 34, 37, 38, 41, 44, 45, 49, 50, 52, 53, 54, 55, 58, 60, 61, 67, 69, 72, 73, 75, 78, 80, 81, 84, 85</td>
</tr>
<tr>
<td>SNPGDSFileClass-class</td>
<td>snpgdsFst, 26, 35, 55, 80</td>
</tr>
<tr>
<td>snpgdsGDS2BED</td>
<td>28, 31, 77</td>
</tr>
<tr>
<td>snpgdsGDS2Eigen</td>
<td>29</td>
</tr>
<tr>
<td>snpgdsGDS2PED</td>
<td>11, 28, 30, 30, 77</td>
</tr>
<tr>
<td>snpgdsGEN2GDS</td>
<td>31</td>
</tr>
<tr>
<td>snpgdsGetGeno</td>
<td>33</td>
</tr>
<tr>
<td>snpgdsIBDMLE</td>
<td>34, 55</td>
</tr>
<tr>
<td>snpgdsIBDMLELogLik</td>
<td>36, 39, 77</td>
</tr>
<tr>
<td>snpgdsIBDSelection</td>
<td>39, 42, 47</td>
</tr>
<tr>
<td>snpgdsIBDMLE</td>
<td>46, 48, 63, 65</td>
</tr>
<tr>
<td>snpgdsIBDMLELogLik</td>
<td>42, 43, 46, 63, 65</td>
</tr>
<tr>
<td>snpgdsIBD pruning</td>
<td>46</td>
</tr>
<tr>
<td>snpgdsIBDSelection</td>
<td>47</td>
</tr>
<tr>
<td>snpgdsIBS</td>
<td>18, 36, 49, 51, 68</td>
</tr>
<tr>
<td>snpgdsIBSNum</td>
<td>30, 50</td>
</tr>
<tr>
<td>snpgdsIndInb</td>
<td>35, 51, 55</td>
</tr>
<tr>
<td>snpgdsIndInbCoef</td>
<td>53</td>
</tr>
<tr>
<td>snpgdsIndivBeta</td>
<td>54</td>
</tr>
<tr>
<td>snpgdsLDMat</td>
<td>55, 58, 59</td>
</tr>
<tr>
<td>snpgdsLDpair</td>
<td>56, 57, 59</td>
</tr>
<tr>
<td>snpgdsLDpruning</td>
<td>56, 58, 58, 79</td>
</tr>
<tr>
<td>snpgdsOpen</td>
<td>12, 26, 60</td>
</tr>
<tr>
<td>snpgdsOption</td>
<td>10, 11, 61, 90, 91</td>
</tr>
<tr>
<td>snpgdsPairIBD</td>
<td>62, 65</td>
</tr>
<tr>
<td>snpgdsPairIBDMLELogLik</td>
<td>63, 64</td>
</tr>
<tr>
<td>snpgdsPairScore</td>
<td>66</td>
</tr>
<tr>
<td>snpgdsPCA</td>
<td>6, 23, 69, 72–76</td>
</tr>
<tr>
<td>snpgdsPCACorr</td>
<td>70, 72, 74, 76</td>
</tr>
</tbody>
</table>
snpqdsPCASampLoading, 70, 73, 73, 76
snpqdsPCASNPLoading, 70, 73, 74, 75
snpqdsPED2GDS, 11, 76
snpqdsSampMissRate, 77, 79, 85
snpqdsSelectSNP, 78
snpqdsSlidingWindow, 79
snpqdsSNPList, 81, 82–84
snpqdsSNPListClass, 13, 82, 82
snpqdsSNPListIntersect, 81, 82, 82
snpqdsSNPListStrand, 81, 83, 83, 84
snpqdsSNPRateFreq, 37, 78, 79, 84
snpqdsSummary, 85
snpqdsTranspose, 86
snpqdsVCF2GDS, 32, 87
snpqdsVCF2GDS_R, 90, 91
SNPRelate (SNPRelate-package), 3
SNPRelate-package, 3