

# Package ‘SNPRelate’

April 15, 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
snpGdsIBSNum . . . . .	50
snpGdsIndInb . . . . .	51
snpGdsIndInbCoef . . . . .	53
snpGdsIndivBeta . . . . .	54
snpGdsLDMat . . . . .	55
snpGdsLDpair . . . . .	57
snpGdsLDpruning . . . . .	58
snpGdsOpen . . . . .	60
snpGdsOption . . . . .	61
snpGdsPairIBD . . . . .	62
snpGdsPairIBDMLELogLik . . . . .	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
snpgdsVCF2GDS_R . . . . .	90

<b>Index</b>	<b>92</b>
--------------	-----------

---

SNPRelate-package	<i>Parallel Computing Toolset for Genome-Wide Association Studies</i>
-------------------	---

---

## Description

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.

## Details

Package:	SNPRelate
Type:	Package
License:	GPL version 3
Depends:	gdsfmt (>= 1.0.4)

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.

Webpage: <http://github.com/zhengxwen/SNPRelate>, <http://corearray.sourceforge.net/>

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

## Author(s)

Xiuwen Zheng <[zhengxwen@gmail.com](mailto:zhengxwen@gmail.com)>

## References

Zheng X, Levine D, Shen J, Gogarten SM, Laurie C, Weir BS. A High-performance Computing Toolset for Relatedness and Principal Component Analysis of SNP Data. *Bioinformatics* (2012); doi: 10.1093/bioinformatics/bts610

## 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(snpgdsExampleFileName())

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
#
```

```

# 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\_genotype

*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_genotype
```

### 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 <code>snpGdsEigMixClass</code> from <a href="#">snpGdsEIGMIX</a> , or an object of <code>snpGdsPCAClass</code> from <a href="#">snpGdsPCA</a>
groups	a list of sample IDs, such like <code>groups = list(CEU = c("NA0101", "NA1022", ...), YRI</code>
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:

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

**Author(s)**

Xiuwen Zheng

**References**

Zheng X, Weir BS. Eigenanalysis on SNP Data with an Interpretation of Identity by Descent. *Theoretical Population Biology*. 2015 Oct 23. pii: S0040-5809(15)00089-1. doi: 10.1016/j.tpb.2015.09.004. [Epub ahead of print]

**See Also**

[snpGdsEIGMIX](#), [snpGdsPCA](#)

**Examples**

```

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

mean(c(z))
mean(diag(z))

```

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

---

```
snpgdsAlleleSwitch     Allele-switching
```

---

## Description

Switch alleles according to the reference if needed.

## Usage

```
snpgdsAlleleSwitch(gdsobj, A.allele, verbose=TRUE)
```

## Arguments

gdsobj	an object of class <code>SNPGDSFileClass</code> , 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

```
# 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)
```

---

snpGDSApartSelection *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

```
snpGDSApartSelection(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

[snpGDSLdPruning](#)

**Examples**

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

snpgdsBED2GDS

*Conversion from PLINK BED to GDS***Description**

Convert a PLINK binary ped file to a GDS file.

**Usage**

```
snpgdsBED2GDS(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 <a href="#">snpgdsOption</a> , 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
snpGSDSClose(genofile)

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

---

snpgdsClose	<i>Close the SNP GDS File</i>
-------------	-------------------------------

---

### Description

Close the SNP GDS file

### Usage

```
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

```
# 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

```
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 <a href="#">snpGDSListClass</a> 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**

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

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

snpgdsSummary("test.gds")
```

---

snpgdsCreateGeno	<i>Create a SNP genotype dataset from a matrix</i>
------------------	--

---

**Description**

To create a GDS file of genotypes from a matrix.

**Usage**

```
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**

<code>gds.fn</code>	the file name of gds
<code>genmat</code>	a matrix of genotypes
<code>sample.id</code>	the sample ids, which should be unique
<code>snp.id</code>	the SNP ids, which should be unique
<code>snp.rs.id</code>	the rs ids for SNPs, which can be not unique
<code>snp.chromosome</code>	the chromosome indices
<code>snp.position</code>	the SNP positions in basepair
<code>snp.allele</code>	the reference/non-reference alleles
<code>snpfirstdim</code>	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)
<code>compress.annotation</code>	the compression method for the variables except genotype
<code>compress.geno</code>	the compression method for the variable genotype
<code>other.vars</code>	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.

**Value**

None.

**Author(s)**

Xiuwen Zheng

**See Also**[snpgdsCreateGenoSet](#), [snpgdsCombineGeno](#)**Examples**

```
# 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)
```

---

snpgdsCreateGenoSet     *Create a SNP genotype dataset from a GDS file*

---

**Description**

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

**Usage**

```
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 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**

[snpgdsCreateGeno](#), [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 279 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%) }
```



```

# |--+ 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
snpGDSClose(gfile)

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

```

---

snpGDSCutTree

*Determine clusters of individuals*


---

### Description

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

### Usage

```

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 <a href="#">snpGDSHCluster</a>
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
dmat	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**

[snpgdsHCluster](#), [snpgdsDrawTree](#), [snpgdsIBS](#), [snpgdsDiss](#)

**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)

#####
# cluster individuals
#

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

# 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"))
```

```
#####
# or cluster individuals by ethnic information
#

rv2 <- snpGDSCutTree(hc, samp.group=pop.group)

# cluster individuals by Z score, specifying 'clust.count'
snpGDSDrawTree(rv2, rv$clust.count, main="HapMap Phase II",
  edgePar = list(col=rgb(0.5,0.5,0.5, 0.75), t.col="black"),
  labels = c("YRI", "CHB/JPT", "CEU"), y.label=0.1)
legend("bottomleft", legend=levels(pop.group), col=1:nlevels(pop.group),
  pch=19, ncol=4, bg="white")

#####
# zoom in ...
#

snpGDSDrawTree(rv2, rv$clust.count, dend.idx = c(1),
  main="HapMap Phase II -- YRI",
  edgePar=list(col=rgb(0.5,0.5,0.5, 0.75), t.col="black"),
  y.label.kinship=TRUE)

snpGDSDrawTree(rv2, rv$clust.count, dend.idx = c(2,2),
  main="HapMap Phase II -- CEU",
  edgePar=list(col=rgb(0.5,0.5,0.5, 0.75), t.col="black"),
  y.label.kinship=TRUE)

snpGDSDrawTree(rv2, rv$clust.count, dend.idx = c(2,1),
  main="HapMap Phase II -- CHB/JPT",
  edgePar=list(col=rgb(0.5,0.5,0.5, 0.75), t.col="black"),
  y.label.kinship=TRUE)
```

---

snpGDSdiss

*Individual dissimilarity analysis*


---

## Description

Calculate the individual dissimilarities for each pair of individuals.

## Usage

```
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

<code>gdsobj</code>	an object of class <code>SNPGDSFileClass</code> , a SNP GDS file
<code>sample.id</code>	a vector of sample id specifying selected samples; if <code>NULL</code> , all samples are used
<code>snp.id</code>	a vector of snp id specifying selected SNPs; if <code>NULL</code> , all SNPs are used

<code>autosome.only</code>	if TRUE, use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
<code>remove.monosnp</code>	if TRUE, remove monomorphic SNPs
<code>maf</code>	to use the SNPs with " <code>&gt;= maf</code> " only; if NaN, no MAF threshold
<code>missing.rate</code>	to use the SNPs with " <code>&lt;= missing.rate</code> " only; if NaN, no missing threshold
<code>num.thread</code>	the number of (CPU) cores used; if NA, detect the number of cores automatically
<code>verbose</code>	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 "snpGDS":

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

### Author(s)

Xiuwen Zheng

### References

Zheng, Xiuwen. 2013. Statistical Prediction of HLA Alleles and Relatedness Analysis in Genome-Wide Association Studies. PhD dissertation, the department of Biostatistics, University of Washington.

Weir BS, Zheng X. SNPs and SNVs in Forensic Science. 2015. Forensic Science International: Genetics Supplement Series.

### See Also

[snpGDS](#)

### 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 <- snpGDS(diss(genofile))
hc <- snpGDS(hc(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 <a href="#">snpGDSCutTree</a>
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). "textlike" 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.

**Details**

The details will be described in future.

**Value**

None.

**Author(s)**

Xiuwen Zheng

**See Also**

[snpgdsCutTree](#)

**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"))
```

---

snpgdsEIGMIX

*Eigen-analysis on SNP genotype data*

---

**Description**

Eigen-analysis on IBD matrix based SNP genotypes.

**Usage**

```
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)
```

**Arguments**

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

**Value**

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

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

**Author(s)**

Xiuwen Zheng

**References**

Zheng X, Weir BS. Eigenanalysis on SNP Data with an Interpretation of Identity by Descent. *Theoretical Population Biology*. 2016 Feb;107:65-76. doi: 10.1016/j.tpb.2015.09.004

**See Also**

[snpGDS EIGMIX](#), [snpGDS PCA](#)

**Examples**

```
# 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 <- snpGDS_EIGMIX(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 <- snpGDS_AdmixProp(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 <- snpGDS_EIGMIX(genofile, sample.id=samp.id[pop_code=="JPT"],
  need.ibdmat=TRUE)
z <- RV$ibdmat

mean(c(z))
mean(diag(z))

# close the genotype file
snpGDS_Close(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()

---

SNPGDSFileClass	<i>SNPGDSFileClass</i>
-----------------	------------------------

---

### 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(snpgdsExampleFileName())
genofile

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

# close the file
snpgdsClose(genofile)
```

---

snpgdsFst	<i>F-statistics (fixation indices)</i>
-----------	--

---

### 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

<code>gdsobj</code>	an object of class <a href="#">SNPGDSFileClass</a> , a SNP GDS file
<code>population</code>	a factor, indicating population information for each individual
<code>method</code>	"W&H02" – relative beta estimator in Weir&Hill 2002, "W&C84" – Fst estimator in Weir&Cockerham 1984
<code>sample.id</code>	a vector of sample id specifying selected samples; if NULL, all samples are used
<code>snp.id</code>	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
with.id	if TRUE, the returned value with sample.id and sample.id
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`.

### 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

### Author(s)

Xiuwen Zheng

### References

Weir, B. S. & Cockerham, C. C. Estimating F-statistics for the analysis of population structure. (1984).

Weir, B. S. & Hill, W. G. Estimating F-statistics. Annual review of genetics 36, 721-50 (2002).

### Examples

```
# 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)
```

snpgdsGDS2BED

*Conversion from GDS to PLINK BED***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 <code>SNPGDSFileClass</code> , 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 <code>NULL</code> , all samples are used
snp.id	a vector of snp id specifying selected SNPs; if <code>NULL</code> , all SNPs are used
snpfirstdim	if <code>TRUE</code> , 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 <code>NULL</code> , determine automatically
verbose	if <code>TRUE</code> , 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](#)

**Examples**

```
# open an example dataset (HapMap)
genofile <- snpGDSOpen(snpGDSExampleFileName())

snpset <- snpGDSSelectSNP(genofile, missing.rate=0.95)
snpGDS2BED(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)
```

---

snpGDS2Eigen

*Conversion from GDS to Eigen (EIGENSTRAT)*


---

**Description**

Convert a GDS file to an EIGENSTRAT file.

**Usage**

```
snpGDS2Eigen(gdsobj, eigen.fn, sample.id=NULL, snp.id=NULL, verbose=TRUE)
```

**Arguments**

<code>gdsobj</code>	an object of class <code>SnpGDSfileClass</code> , a SNP GDS file
<code>eigen.fn</code>	the file name of EIGENSTRAT
<code>sample.id</code>	a vector of sample id specifying selected samples; if <code>NULL</code> , all samples are used
<code>snp.id</code>	a vector of snp id specifying selected SNPs; if <code>NULL</code> , all SNPs are used
<code>verbose</code>	if <code>TRUE</code> , 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

Patterson N, Price AL, Reich D (2006) Population structure and eigenanalysis. *PLoS Genetics* 2:e190.

Price AL, Patterson NJ, Plenge RM, Weinblatt ME, Shadick NA, Reich D (2006) Principal components analysis corrects for stratification in genome-wide association studies. *Nat Genet.* 38, 904-909.

<http://corearray.sourceforge.net/>

## See Also

[snpgdsGDS2PED](#)

## Examples

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

snpsset <- snpgdsSelectSNP(genofile, missing.rate=0.95)
snpgdsGDS2Eigen(genofile, eigen.fn="tmpeigen", snp.id=snpsset)

# close the genotype file
snpgdsClose(genofile)

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

---

snpgdsGDS2PED	<i>Conversion from GDS to PED</i>
---------------	-----------------------------------

---

## 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 ( <a href="#">gds.class</a> )
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

**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**

[snpgdsGDS2BED](#)

**Examples**

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

# GDS ==> PED
snpgdsGDS2PED(genofile, ped.fn="tmp")

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

---

snpgdsGEN2GDS

*Conversion from Oxford GEN format to GDS*

---

**Description**

Convert an Oxford GEN file (text format) to a GDS file.

**Usage**

```
snpgdsGEN2GDS(gen.fn, sample.fn, out.fn, chr.code=NULL,
  call.threshold=0.9, version=c(">=2.0", "<=1.1.5"),
  snpfirstdim=FALSE, compress.annotation="ZIP_RA.max", compress.geno="",
  verbose=TRUE)
```

**Arguments**

<code>gen.fn</code>	the file name of Oxford GEN text file(s), it could be a vector indicate merging all files
<code>sample.fn</code>	the file name of sample annotation
<code>out.fn</code>	the output GDS file
<code>chr.code</code>	a vector of chromosome code according to <code>gen.fn</code> , indicating chromosomes. It could be either numeric or character-type
<code>call.threshold</code>	the threshold to determine missing genotypes
<code>version</code>	either " <code>&gt;=2.0</code> " or " <code>&lt;=1.1.5</code> ", see details
<code>snpfirstdim</code>	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)
<code>compress.annotation</code>	the compression method for the GDS variables, except "genotype"; optional values are defined in the function <code>add.gdsn</code>
<code>compress.geno</code>	the compression method for "genotype"; optional values are defined in the function <code>add.gdsn</code>
<code>verbose</code>	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](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 <a href="#">SNPGDSFileClass</a> , 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

```

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

<code>gdsobj</code>	an object of class <code>SNPGDSFileClass</code> , a SNP GDS file
<code>sample.id</code>	a vector of sample id specifying selected samples; if <code>NULL</code> , all samples are used
<code>snp.id</code>	a vector of snp id specifying selected SNPs; if <code>NULL</code> , all SNPs are used
<code>autosome.only</code>	if <code>TRUE</code> , use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
<code>remove.monosnp</code>	if <code>TRUE</code> , remove monomorphic SNPs
<code>maf</code>	to use the SNPs with " <code>&gt;= maf</code> " only; if <code>NaN</code> , no MAF threshold
<code>missing.rate</code>	to use the SNPs with " <code>&lt;= missing.rate</code> " only; if <code>NaN</code> , no missing threshold
<code>method</code>	"Visscher" – genetic relationship matrix defined in CGTA; "Eigenstrat" – genetic covariance matrix in EIGENSTRAT; "EIGMIX" – coancestry matrix defined in Zheng & Weir (2015), "W&Z15" – individual beta estimator
<code>num.thread</code>	the number of (CPU) cores used; if <code>NA</code> , detect the number of cores automatically
<code>with.id</code>	if <code>TRUE</code> , the returned value with <code>sample.id</code> and <code>snp.id</code>
<code>verbose</code>	if <code>TRUE</code> , show information

## Value

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

<code>sample.id</code>	the sample ids used in the analysis
<code>snp.id</code>	the SNP ids used in the analysis
<code>grm</code>	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.

**Author(s)**

Xiuwen Zheng

**References**

Patterson, N., Price, A. L. & Reich, D. Population structure and eigenanalysis. *PLoS Genet.* 2, e190 (2006).

Yang, J., Lee, S. H., Goddard, M. E. & Visscher, P. M. GCTA: a tool for genome-wide complex trait analysis. *American journal of human genetics* 88, 76-82 (2011).

Zheng X, Weir BS. Eigenanalysis on SNP Data with an Interpretation of Identity by Descent. *Theoretical Population Biology.* 2016 Feb;107:65-76. doi: 10.1016/j.tpb.2015.09.004

Weir BS, Zheng X. SNPs and SNVs in Forensic Science. *Forensic Science International: Genetics Supplement Series.* 2015. doi:10.1016/j.fsigss.2015.09.106

**See Also**

[snpGdsIndInb](#), [snpGdsFst](#)

**Examples**

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

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

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

# close the file
snpGdsClose(genofile)
```

---

snpGdsHCluster

*Hierarchical cluster analysis*

---

**Description**

Perform hierarchical cluster analysis on the dissimilarity matrix.

**Usage**

```
snpGdsHCluster(dist, sample.id=NULL, need.mat=TRUE, hang=0.25)
```

**Arguments**

<code>dist</code>	an object of "snpgdsDissClass" from <a href="#">snpgdsDiss</a> , an object of "snpgdsIBSClass" from <a href="#">snpgdsIBS</a> , or a square matrix for dissimilarity
<code>sample.id</code>	to specify sample id, only work if <code>dist</code> is a matrix
<code>need.mat</code>	if TRUE, store the dissimilarity matrix in the result
<code>hang</code>	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"):

<code>sample.id</code>	the sample ids used in the analysis
<code>hclust</code>	an object returned from <a href="#">hclust</a>
<code>dendrogram</code>	
<code>dist</code>	the dissimilarity matrix, if <code>need.mat = TRUE</code>

**Author(s)**

Xiuwen Zheng

**See Also**

[snpgdsIBS](#), [snpgdsDiss](#), [snpgdsCutTree](#)

**Examples**

```
# 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 <a href="#">SNPGDSFileClass</a> , 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

Wigginton, J. E., Cutler, D. J. & Abecasis, G. R. A note on exact tests of Hardy-Weinberg equilibrium. *Am. J. Hum. Genet.* 76, 887-93 (2005).

## See Also

[snpgdsSNPRateFreq](#)

## Examples

```
# 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

```

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

<code>gdsobj</code>	an object of class <code>SNPGDSFileClass</code> , a SNP GDS file
<code>sample.id</code>	a vector of sample id specifying selected samples; if <code>NULL</code> , all samples are used
<code>snp.id</code>	a vector of snp id specifying selected SNPs; if <code>NULL</code> , all SNPs are used
<code>autosome.only</code>	if <code>TRUE</code> , use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
<code>remove.monosnp</code>	if <code>TRUE</code> , remove monomorphic SNPs
<code>maf</code>	to use the SNPs with " <code>&gt;= maf</code> " only; if <code>NaN</code> , no MAF threshold
<code>missing.rate</code>	to use the SNPs with " <code>&lt;= missing.rate</code> " only; if <code>NaN</code> , no missing threshold
<code>type</code>	"KING-robust" – relationship inference in the presence of population stratification; "KING-homo" – relationship inference in a homogeneous population
<code>family.id</code>	if <code>NULL</code> , all individuals are treated as singletons; if family id is given, within- and between-family relationship are estimated differently. If <code>sample.id=NULL</code> , <code>family.id</code> should have the same length as "sample.id" in the GDS file, otherwise <code>family.id</code> should have the same length and order as the argument <code>sample.id</code>
<code>num.thread</code>	the number of (CPU) cores used; if <code>NA</code> , detect the number of cores automatically
<code>verbose</code>	if <code>TRUE</code> , 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 (IBS<sub>0</sub>) and kinship coefficient (kinship). With "KING-homo" it would return the probability of sharing one IBD (k<sub>1</sub>) and the probability of sharing zero IBD (k<sub>0</sub>).

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:

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

## Author(s)

Xiuwen Zheng

## References

Manichaikul A, Mychaleckyj JC, Rich SS, Daly K, Sale M, Chen WM. Robust relationship inference in genome-wide association studies. *Bioinformatics*. 2010 Nov 15;26(22):2867-73.

## See Also

[snpGDSIBDMLE](#), [snpGDSIBDMoM](#)

## Examples

```
# open an example dataset (HapMap)
genofile <- snpGDSOpen(snpGDSExampleFileName())

# 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**

```
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**

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

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

### Author(s)

Xiuwen Zheng

### References

- Milligan BG. 2003. Maximum-likelihood estimation of relatedness. *Genetics* 163:1153-1167.
- Weir BS, Anderson AD, Hepler AB. 2006. Genetic relatedness analysis: modern data and new challenges. *Nat Rev Genet.* 7(10):771-80.
- Choi Y, Wijsman EM, Weir BS. 2009. Case-control association testing in the presence of unknown relationships. *Genet Epidemiol* 33(8):668-78.
- Jacquard, A. *Structures Genetiques des Populations* (Masson & Cie, Paris, 1970); English translation available in Charlesworth, D. & Charlesworth, B. *Genetics of Human Populations* (Springer, New York, 1974).

### See Also

[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"]
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
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 <- snpGDSRateFreq(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	<i>Log likelihood for MLE method in the Identity-By-Descent (IBD) Analysis</i>
--------------------	--

---

## 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 <code>SNPGDSfileClass</code> , a SNP GDS file
ibdoobj	the <code>snpgdsIBDClass</code> object returned from <code>snpgdsIBDMLE</code>
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 `ibdoobj`. \ 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**

- Milligan BG. 2003. Maximum-likelihood estimation of relatedness. *Genetics* 163:1153-1167.
- Weir BS, Anderson AD, Hepler AB. 2006. Genetic relatedness analysis: modern data and new challenges. *Nat Rev Genet.* 7(10):771-80.
- Choi Y, Wijsman EM, Weir BS. 2009. Case-control association testing in the presence of unknown relationships. *Genet Epidemiol* 33(8):668-78.

**See Also**

[snpgdsIBDMLE](#), [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"]
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
```

```

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)

```

---

snpGDSIBDMoM	<i>PLINK method of moment (MoM) for the Identity-By-Descent (IBD) Analysis</i>
--------------	--

---

## Description

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

## Usage

```

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

<code>gdsobj</code>	an object of class <code>SNPGDSfileClass</code> , a SNP GDS file
<code>sample.id</code>	a vector of sample id specifying selected samples; if <code>NULL</code> , all samples are used
<code>snp.id</code>	a vector of snp id specifying selected SNPs; if <code>NULL</code> , 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 genealogical region ( $k_0 k_1 \geq k_2^2$ )
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
$k_0$	IBD coefficient, the probability of sharing ZERO IBD
$k_1$	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

[snpGdsIBDMLE](#), [snpGdsIBDMLELogLik](#)

**Examples**

```

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

#####
# 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)
names(pibd)

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)

```

**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 <a href="#">snpgdsIBDMLE</a> or <a href="#">snpgdsIBDMoM</a>
kinship.cutoff	select the individual pairs with kinship coefficients $\geq$ 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
```



```
# 1 NA19152 NA19154 0.010749154 0.9892508 0.24731271
# 2 NA19152 NA19093 0.848207777 0.1517922 0.03794806
# 3 NA19139 NA19138 0.010788047 0.9770181 0.25035144
# 4 NA19139 NA19137 0.012900661 0.9870993 0.24677483
# 5 NA18912 NA18914 0.008633077 0.9913669 0.24784173
# 6 NA19160 NA19161 0.008635754 0.9847777 0.24948770
```

---

snpGDSIBS

*Identity-By-State (IBS) proportion*


---

## Description

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

## Usage

```
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

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

<code>sample.id</code>	the sample ids used in the analysis
<code>snp.id</code>	the SNP ids used in the analysis
<code>ibs</code>	a matrix of IBS proportion, "# of samples" x "# of samples"

## Author(s)

Xiuwen Zheng

**See Also**

[snpgdsIBSNum](#)

**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)
```

---

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**

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

**Value**

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

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

**Author(s)**

Xiuwen Zheng

**See Also**

[snpGdsIBS](#)

**Examples**

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

RV <- snpGdsIBSNum(genofile)
pop <- read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))
L <- order(pop)
image(RV$ibs0[L, L]/length(RV$snp.id))

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

---

snpGdsIndInb

*Individual Inbreeding Coefficients*

---

**Description**

To calculate individual inbreeding coefficients using SNP genotype data

**Usage**

```
snpGdsIndInb(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**

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

Yang J, Benyamin B, McEvoy BP, Gordon S, Henders AK, Nyholt DR, Madden PA, Heath AC, Martin NG, Montgomery GW, Goddard ME, Visscher PM. 2010. Common SNPs explain a large proportion of the heritability for human height. *Nat Genet.* 42(7):565-9. Epub 2010 Jun 20.

**Examples**

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

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

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

---

snpgdsIndInbCoef      *Individual Inbreeding Coefficient*

---

### Description

To calculate an individual inbreeding coefficient using SNP genotype data

### Usage

```
snpgdsIndInbCoef(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
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.

### 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

Yang J, Benyamin B, McEvoy BP, Gordon S, Henders AK, Nyholt DR, Madden PA, Heath AC, Martin NG, Montgomery GW, Goddard ME, Visscher PM. 2010. Common SNPs explain a large proportion of the heritability for human height. *Nat Genet.* 42(7):565-9. Epub 2010 Jun 20.

### Examples

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

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))
```

```

snpGDSIndivBeta(g[chr1idx], AF$AlleleFreq[chr1idx], method="mom.weir")
snpGDSIndivBeta(g[chr1idx], AF$AlleleFreq[chr1idx], method="mom.visscher")
snpGDSIndivBeta(g[chr1idx], AF$AlleleFreq[chr1idx], method="mle")

# close the genotype file
snpGDSClose(genofile)

```

---

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

**Author(s)**

Xiuwen Zheng

**References**

Weir BS, Zheng X. SNPs and SNVs in Forensic Science. Forensic Science International: Genetics Supplement Series. 2015. doi:10.1016/j.fsigss.2015.09.106

**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
snpGSDSClose(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**

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

<code>sample.id</code>	the sample ids used in the analysis
<code>snp.id</code>	the SNP ids used in the analysis
<code>LD</code>	a matrix of LD values
<code>slide</code>	the size of sliding window

**Author(s)**

Xiuwen Zheng

**References**

- Weir B: Inferences about linkage disequilibrium. *Biometrics* 1979; 35: 235-254.
- Weir B: *Genetic Data Analysis II*. Sunderland, MA: Sinauer Associates, 1996.
- Weir BS, Cockerham CC: Complete characterization of disequilibrium at two loci; in Feldman MW (ed): *Mathematical Evolutionary Theory*. Princeton, NJ: Princeton University Press, 1989.

**See Also**

[snpGdsLDpair](#), [snpGdsLDpruning](#)

**Examples**

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

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

# 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

```
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
if method = "r" or "dprime",	
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

Weir B: Inferences about linkage disequilibrium. *Biometrics* 1979; 35: 235-254.  
 Weir B: *Genetic Data Analysis II*. Sunderland, MA: Sinauer Associates, 1996.  
 Weir BS, Cockerham CC: Complete characterization of disequilibrium at two loci; in Feldman MW (ed): *Mathematical Evolutionary Theory*. Princeton, NJ: Princeton University Press, 1989.

**See Also**

[snpgdsLDMat](#), [snpgdsLDpruning](#)

**Examples**

```
# 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)
```

---

snpgdsLDpruning

*Linkage Disequilibrium (LD) based SNP pruning*

---

**Description**

Recursively removes SNPs within a sliding window

**Usage**

```
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**

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

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

- Weir B: Inferences about linkage disequilibrium. *Biometrics* 1979; 35: 235-254.
- Weir B: *Genetic Data Analysis II*. Sunderland, MA: Sinauer Associates, 1996.
- Weir BS, Cockerham CC: Complete characterization of disequilibrium at two loci; in Feldman MW (ed): *Mathematical Evolutionary Theory*. Princeton, NJ: Princeton University Press, 1989.

## See Also

[snpGdsLDMat](#), [snpGdsLDpair](#)

## Examples

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

set.seed(1000)
snpset <- snpGdsLDpruning(genofile)
names(snpset)
```

```

# [1] "chr1" "chr2" "chr3" "chr4" "chr5" "chr6" "chr7" "chr8" "chr9"
# [10] "chr10" "chr11" "chr12" "chr13" "chr14" "chr15" "chr16" "chr17" "chr18"
# .....
head(snpset$chr1)
# [1] 1 2 3 4 5 6

# get SNP ids
snp.id <- unlist(snpset)

# close the genotype file
snpgdsClose(genofile)

```

---

snpgdsOpen

*Open a SNP GDS File*


---

### Description

Open a SNP GDS file

### Usage

```
snpgdsOpen(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 <a href="#">openfn.gds</a>
allow.fork	TRUE for parallel environment using forking, see <a href="#">openfn.gds</a>

### 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

[snpgdsClose](#)

**Examples**

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

genofile

# close the file
snpgdsClose(genofile)
```

---

snpgdsOption	<i>Option settings: chromosome coding, etc</i>
--------------	--

---

**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 <a href="#">SNPGDSFileClass</a> , 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)
```

---

snpGdsPairIBD

*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

```
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 ( $k_0 k_1 \geq k_2^2$ )
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 $\text{reltol} * (\text{abs}(\log \text{likelihood with the initial parameters}) + \text{reltol})$ at a step.
coeff.correct	TRUE by default, see details
out.num.iter	if TRUE, output the numbers of iterations
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 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.

**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**

- Milligan BG. 2003. Maximum-likelihood estimation of relatedness. *Genetics* 163:1153-1167.
- Weir BS, Anderson AD, Hepler AB. 2006. Genetic relatedness analysis: modern data and new challenges. *Nat Rev Genet.* 7(10):771-80.
- Choi Y, Wijsman EM, Weir BS. 2009. Case-control association testing in the presence of unknown relationships. *Genet Epidemiol* 33(8):668-78.
- 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**

[snpGDS::PairIBDMLELogLik](#), [snpGDS::IBDMLE](#), [snpGDS::IBDMLELogLik](#), [snpGDS::IBDMoM](#)

**Examples**

```
# open an example dataset (HapMap)
genofile <- snpGDS::Open(snpGDS::ExampleFileName())

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 <- snpGDS::LDPruning(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 <- snpGDS::SNPRateFreq(genofile, sample.id=YRI.id, snp.id=snpset,
  with.id=TRUE)
summary(RF$AlleleFreq)

subMLE <- snpGDS::IBDMLE(genofile, sample.id=YRI.id[1:n], snp.id=RF$snp.id,
  allele.freq=RF$AlleleFreq)
subMoM <- snpGDS::IBDMoM(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$AlleleFreq, "EM"))
  print(snpGdsPairIBDMLELogLik(mat[,1], mat[,i], RF$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)

```

---

snpGdsPairIBDMLELogLik

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

---

## Description

Calculate the log likelihood values from maximum likelihood estimation.

## Usage

```

snpGdsPairIBDMLELogLik(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



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

- Milligan BG. 2003. Maximum-likelihood estimation of relatedness. *Genetics* 163:1153-1167.
- Weir BS, Anderson AD, Hepler AB. 2006. Genetic relatedness analysis: modern data and new challenges. *Nat Rev Genet.* 7(10):771-80.
- Choi Y, Wijsman EM, Weir BS. 2009. Case-control association testing in the presence of unknown relationships. *Genet Epidemiol* 33(8):668-78.

### See Also

[snpGdsPairIBD](#), [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$AlleleFreq, "EM"))
  print(snpGdsPairIBDMLELogLik(mat[,1], mat[,i], RF$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)

```

---

snpGdsPairScore

*Genotype Score for Pairs of Individuals*


---

### Description

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

### Usage

```

snpGdsPairScore(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**

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

**Details**

Patient (sample1.id)	Coded Genotype	Donor (sample2.id)	Coded Genotype	IBS	GVH	HVG
AA	0	AA	0	2	0	0
AA	0	AB	1	1	0	1
AA	0	BB	2	0	2	2
AB	1	AA	0	1	1	0
AB	1	AB	1	2	0	0
AB	1	BB	2	1	1	0
BB	2	AA	0	0	2	2
BB	2	AB	1	1	0	1
BB	2	BB	2	2	0	0

**Value**

Return a list:

<code>sample.id</code>	the sample ids used in the analysis, if <code>with.id=TRUE</code>
<code>snp.id</code>	the SNP ids used in the analysis, if <code>with.id=TRUE</code>
<code>score</code>	a matrix of genotype score: if <code>type="per.pair"</code> , a <code>data.frame</code> 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 <code>type="per.snp"</code> , 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 <code>type="matrix"</code> , a # of pairs-by-# of SNPs matrix with rows for pairs of individuals

**Author(s)**

Xiuwen Zheng

## References

Warren, E. H., Zhang, X. C., Li, S., Fan, W., Storer, B. E., Chien, J. W., Boeckh, M. J., et al. (2012). Effect of MHC and non-MHC donor/recipient genetic disparity on the outcome of allogeneic HCT. *Blood*, 120(14), 2796-806. doi:10.1182/blood-2012-04-347286

## 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,
```

```

    method="IBS", type="gds.file", output="tmp.gds")
(f <- snpGDSOpen("tmp.gds"))
snpGDSClose(f)

# close the file
snpGDSClose(genofile)

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

```

snpGDSPCA

*Principal Component Analysis (PCA) on SNP genotype data***Description**

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

**Usage**

```

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)
## S3 method for class 'snpGDSPCAclass'
plot(x, eig=c(1L,2L), ...)

```

**Arguments**

<code>gdsobj</code>	an object of class <code>SNPGDSFileClass</code> , a SNP GDS file
<code>sample.id</code>	a vector of sample id specifying selected samples; if <code>NULL</code> , all samples are used
<code>snp.id</code>	a vector of snp id specifying selected SNPs; if <code>NULL</code> , all SNPs are used
<code>autosome.only</code>	if <code>TRUE</code> , use autosomal SNPs only; if it is a numeric or character value, keep SNPs according to the specified chromosome
<code>remove.monosnp</code>	if <code>TRUE</code> , remove monomorphic SNPs
<code>maf</code>	to use the SNPs with " <code>&gt;= maf</code> " only; if <code>NaN</code> , no MAF threshold
<code>missing.rate</code>	to use the SNPs with " <code>&lt;= missing.rate</code> " only; if <code>NaN</code> , no missing threshold
<code>eigen.cnt</code>	output the number of eigenvectors; if <code>eigen.cnt &lt;= 0</code> , then return all eigenvectors
<code>algorithm</code>	"exact", traditional exact calculation; "randomized", fast PCA with randomized algorithm introduced in Galinsky et al. 2016
<code>num.thread</code>	the number of (CPU) cores used; if <code>NA</code> , detect the number of cores automatically
<code>bayesian</code>	if <code>TRUE</code> , use bayesian normalization
<code>need.genmat</code>	if <code>TRUE</code> , return the genetic covariance matrix
<code>genmat.only</code>	return the genetic covariance matrix only, do not compute the eigenvalues and eigenvectors

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:

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

### Author(s)

Xiuwen Zheng

### References

Patterson N, Price AL, Reich D. Population structure and eigenanalysis. *PLoS Genet.* 2006 Dec;2(12):e190.

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. *Am J Hum Genet.* 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

### 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 -0.08680864 -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"

```

```

# [4,] "NA19160"      "YRI"
# [5,] "NA07034"      "CEU"
# ...

# make a data.frame
tab <- data.frame(sample.id = RV$sample.id,
  pop = factor(pop_code)[match(RV$sample.id, samp.id)],
  EV1 = RV$eigenvect[,1], # the first eigenvector
  EV2 = RV$eigenvect[,2], # the second eigenvector
  stringsAsFactors = FALSE)
head(tab)
# sample.id pop EV1 EV2
# 1 NA19152 YRI -0.08411287 -0.01226860
# 2 NA19139 YRI -0.08360644 -0.01085849
# 3 NA18912 YRI -0.08110808 -0.01184524
# 4 NA19160 YRI -0.08680864 -0.01447106
# 5 NA07034 CEU 0.03109761 0.07709255
# 6 NA07055 CEU 0.03228450 0.08155730

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

# close the file
snpgdsClose(genofile)

```

---

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 <a href="#">snpgdsPCA</a>
gdsobj	an object of class <a href="#">SNPGDSfileClass</a> , 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



**Value**

Return a list:

sample.id	the sample ids used in the analysis
snp.id	the SNP ids used in the analysis
snpcorr	a matrix of correlation coefficients, "# of eigenvectors" x "# of SNPs"

**Author(s)**

Xiuwen Zheng

**References**

Patterson N, Price AL, Reich D (2006) Population structure and eigenanalysis. *PLoS Genetics* 2:e190.

**See Also**

[snpGDSPCA](#), [snpGDSPCASampLoading](#), [snpGDSPCASNPLoading](#)

**Examples**

```
# 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)
```

---

snpGDSPCASampLoading *Project individuals onto existing principal component axes*

---

**Description**

To calculate the sample eigenvectors using the specified SNP loadings

**Usage**

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

**Arguments**

loadobj	the <code>snpGDSPCASNPLoadingClass</code> object, returned from <a href="#">snpGDSPCASNPLoading</a>
gdsobj	an object of class <code>SNPGDSFileClass</code> , a SNP GDS file
sample.id	a vector of sample id specifying selected samples; if <code>NULL</code> , all samples are used
num.thread	the number of CPU cores used
verbose	if <code>TRUE</code> , show information

**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:

<code>sample.id</code>	the sample ids used in the analysis
<code>snp.id</code>	the SNP ids used in the analysis
<code>eigenval</code>	eigenvalues
<code>eigenvect</code>	eigenvectors, “# of samples” x “eigen.cnt”
<code>TraceXTX</code>	the trace of the genetic covariance matrix
<code>Bayesian</code>	whether use bayesian normalization

**Author(s)**

Xiuwen Zheng

**References**

Patterson N, Price AL, Reich D (2006) Population structure and eigenanalysis. *PLoS Genetics* 2:e190.

Zhu, X., Li, S., Cooper, R. S., and Elston, R. C. (2008). A unified association analysis approach for family and unrelated samples correcting for stratification. *Am J Hum Genet*, 82(2), 352-365.

**See Also**

[snpGdsPCA](#), [snpGdsPCACorr](#), [snpGdsPCASNPLoading](#)

**Examples**

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

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)
```

---

snpGDSPCASNPLoading     *SNP loadings in principal component analysis*

---

### Description

To calculate the SNP loadings in Principal Component Analysis

### Usage

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

### Arguments

pcaobj	the snpGDSPCAClass object returned from the function <a href="#">snpGDSPCA</a>
gdsobj	an object of class <a href="#">SNPGDSFileClass</a> , 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

- Patterson N, Price AL, Reich D (2006) Population structure and eigenanalysis. *PLoS Genetics* 2:e190.
- Price AL, Patterson NJ, Plenge RM, Weinblatt ME, Shadick NA, Reich D (2006) Principal components analysis corrects for stratification in genome-wide association studies. *Nat Genet.* 38, 904-909.
- Zhu, X., Li, S., Cooper, R. S., and Elston, R. C. (2008). A unified association analysis approach for family and unrelated samples correcting for stratification. *Am J Hum Genet*, 82(2), 352-365.

**See Also**

[snpgdsPCA](#), [snpgdsPCASampLoading](#), [snpgdsPCACorr](#)

**Examples**

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

PCARV <- snpgdsPCA(genofile, eigen.cnt=8)
SnpLoad <- snpgdsPCASNPLoading(PCARV, genofile)

names(SnpLoad)
# [1] "sample.id" "snp.id" "eigenval" "snploading" "TraceXTX"
# [6] "Bayesian" "avefreq" "scale"
dim(SnpLoad$snploading)
# [1] 8 8722

plot(SnpLoad$snploading[1,], type="h", ylab="PC 1")

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

---

snpgdsPED2GDS

*Conversion from PLINK PED to GDS*


---

**Description**

Convert a PLINK PED text file to a GDS file.

**Usage**

```
snpgdsPED2GDS(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 <code>add.gdsn</code>
compress.geno	the compression method for "genotype"; optional values are defined in the function <code>add.gdsn</code>
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**

[snpGDS2PED](#), [snpGDSBED2GDS](#), [snpGDS2BED](#)

**Examples**

```
# open
genofile <- snpGDSopen(snpGDSexampleFileName())

snpGDS2PED(genofile, "tmp")

# close
snpGDSclose(genofile)

# PED ==> GDS
snpGDS2GDS("tmp.ped", "tmp.map", "test.gds")

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

---

snpGDSampMissRate      *Missing Rate of Samples*

---

**Description**

Return the missing fraction for each sample

**Usage**

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

**Arguments**

gdsobj	an object of class <code>SNPGDSfileClass</code> , 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**

[snpgdsSNPRateFreq](#)

**Examples**

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

RV <- snpgdsSampMissRate(genofile)
summary(RV)

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

---

snpgdsSelectSNP	<i>SNP selection</i>
-----------------	----------------------

---

**Description**

Create a list of candidate SNPs based on specified criteria

**Usage**

```
snpgdsSelectSNP(gdsobj, sample.id=NULL, snp.id=NULL, autosome.only=TRUE,
  remove.monosnp=TRUE, maf=NaN, missing.rate=NaN, verbose=TRUE)
```

**Arguments**

gdsobj	an object of class <code>SNPGDSfileClass</code> , 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
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 any MAF threshold  
missing.rate to use the SNPs with " $\leq$  missing.rate" only; if NaN, no any missing threshold  
verbose if TRUE, show information

### Value

Return a list of snp ids.

### Author(s)

Xiuwen Zheng

### See Also

[snpGDSsmpMissRate](#), [snpGDSsnprateFreq](#), [snpGDSLDpruning](#)

### Examples

```
# 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)
```

---

snpGDSslidingWindow *Sliding window*

---

### Description

Apply a user-defined function with a sliding window.

### Usage

```
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**

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

**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)
```



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

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

---

snpGDSNPList                      *Create a SNP list object*

---

## Description

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

## Usage

```
snpGDSNPList(gdsobj, sample.id=NULL)
```

## Arguments

gdsobj	an object of class <a href="#">SNPGDSFileClass</a> , 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 `snpGDSNPListClass` 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

[snpGDSNPListIntersect](#), [snpGDSNPListStrand](#)

## Examples

```
# open an example dataset (HapMap)
genofile <- snpGDSOpen(snpGDSExampleFileName())

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

# close the file
snpGDSClose(genofile)
```

---

snpGdsSNPListClass     *the class of a SNP list*

---

### Description

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](#)

---

snpGdsSNPListIntersect  
*Get a common SNP list between two SNP list objects*

---

### Description

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 <a href="#">snpGdsSNPListClass</a>
snplist2	the second SNP list object <a href="#">snpGdsSNPListClass</a>

### 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

**Author(s)**

Xiuwen Zheng

**See Also**[snpGDS\\_SNPList](#), [snpGDS\\_SNPListStrand](#)**Examples**

```
# open an example dataset (HapMap)
genofile <- snpGDS_Open(snpGDS_ExampleFileName())

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

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

summary(snplist$afreq)

# close the file
snpGDS_Close(genofile)
```

---

snpGDS\_SNPListStrand    *Switch allele strand.*

---

**Description**

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

**Usage**

```
snpGDS_SNPListStrand(snplist1, snplist2, same.strand=FALSE)
```

**Arguments**

snplist1	the first SNP list object “snpGDS_SNPListClass”
snplist2	the second SNP list object “snpGDS_SNPListClass”
same.strand	TRUE assuming alleles are on the same strand (e.g., forward strand); otherwise, FALSE not assuming whether on the same strand or not

**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

**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
L <- snpgdsSNPListStrand(snplist1, snplist2)

table(L, exclude=NULL)

# close the file
snpgdsClose(genofile)
```

---

snpgdsSNPRateFreq      *Allele Frequency, Minor Allele Frequency, Missing Rate of SNPs*

---

**Description**

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

**Usage**

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

**Arguments**

gdsobj	an object of class <a href="#">SNPGDSfileClass</a> , 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, return sample and SNP IDs

**Value**

Return a list:

AlleleFreq	allele frequencies
MinorFreq	minor allele frequencies
MissingRate	missing rates
sample.id	sample id, if with.id=TRUE
snp.id	SNP id, if with.id=TRUE

**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	<i>Summary of GDS genotype file</i>
---------------	-------------------------------------

---

**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 <a href="#">SNPGDSFileClass</a>
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	<i>Transpose genotypic matrix</i>
-----------------	-----------------------------------

---

**Description**

Transpose the genotypic matrix if needed.

**Usage**

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

**Arguments**

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

**Value**

None.

**Author(s)**

Xiuwen Zheng

**Examples**

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

# copy the file
file.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
unlink("test.gds", force=TRUE)
```

---

snpgdsVCF2GDS	<i>Reformat VCF file(s)</i>
---------------	-----------------------------

---

**Description**

Reformat Variant Call Format (VCF) file(s)

**Usage**

```
snpgdsVCF2GDS(vcf.fn, out.fn, 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 the 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 auto-some 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

The variant call format and VCFtools. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. *Bioinformatics*. 2011 Aug 1;27(15):2156-8. Epub 2011 Jun 7.

<http://corearray.sourceforge.net/>

### See Also

[snpgdsBED2GDS](#)

### Examples

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

snpgdsVCF2GDS(vcf.fn, "test1.gds", method="biallelic.only")
snpgdsSummary("test1.gds")

snpgdsVCF2GDS(vcf.fn, "test2.gds", method="biallelic.only", snpfirstdim=TRUE)
snpgdsSummary("test2.gds")

snpgdsVCF2GDS(vcf.fn, "test3.gds", method="copy.num.of.ref", snpfirstdim=TRUE)
snpgdsSummary("test3.gds")

snpgdsVCF2GDS(vcf.fn, "test4.gds", method="copy.num.of.ref")
snpgdsSummary("test4.gds")

snpgdsVCF2GDS(vcf.fn, "test5.gds", method="copy.num.of.ref",
  ref.allele=c("A", "T", "T", "T", "A"))
snpgdsSummary("test5.gds")

# open "test1.gds"
(genofile <- snpgdsOpen("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

```
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 <a href="#">snpgdsOption</a> , 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 = "biallelic.only": to exact bi-allelic and polymorphic SNP data; 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**

The variant call format and VCFtools. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. *Bioinformatics*. 2011 Aug 1;27(15):2156-8. Epub 2011 Jun 7.

**See Also**

[snpGDSVCF2GDS\\_R](#), [snpGDSOption](#), [snpGDSBED2GDS](#)

**Examples**

```
# The VCF file
vcf.fn <- system.file("extdata", "sequence.vcf", package="SNPRelate")
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
- 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
- snpgdsIBSNum, 50
- snpgdsIndInb, 51
- snpgdsIndInbCoef, 53
- snpgdsIndivBeta, 54
- snpgdsLDMat, 55
- snpgdsLDpair, 57
- snpgdsLDpruning, 58
- snpgdsOpen, 60
- snpgdsOption, 61
- snpgdsPairIBD, 62
- snpgdsPairIBDMLELogLik, 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
- 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
- snpgdsIBSNum, 50
- snpgdsIndInb, 51
- snpgdsIndInbCoef, 53
- snpgdsIndivBeta, 54
- snpgdsLDMat, 55
- snpgdsLDpair, 57
- snpgdsLDpruning, 58
- snpgdsOpen, 60
- snpgdsOption, 61
- snpgdsPairIBD, 62
- snpgdsPairIBDMLELogLik, 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
- snpgdsVCF2GDS\_R, 90
- SNPRelate-package, 3
- \*Topic **PCA**
  - snpgdsPCA, 69
  - snpgdsPCACorr, 72
  - snpgdsPCASampLoading, 73
  - snpgdsPCASNPLoading, 75
- \*Topic **datasets**
  - hapmap\_geno, 5
- \*Topic **gds**
  - snpgdsVCF2GDS\_R, 90
- cleanup.gds, 86
- closefn.gds, 12
- gds.class, 26, 30
- gdsfmt, 11, 28, 29, 31, 32, 77, 87, 90
- hapmap\_geno, 5
- hclust, 36
- openfn.gds, 60
- plot.snpgdsPCAClass (snpgdsPCA), 69
- snpgdsAdmixProp, 6
- snpgdsAlleleSwitch, 8
- snpgdsApartSelection, 9
- snpgdsBED2GDS, 10, 28, 32, 77, 88, 91
- snpgdsClose, 12, 26, 60
- snpgdsCombineGeno, 13, 15, 16
- snpgdsCreateGeno, 13, 14, 16
- snpgdsCreateGenoSet, 13, 15, 15
- snpgdsCutTree, 17, 21, 22, 36
- snpgdsDiss, 18, 19, 36
- snpgdsDrawTree, 18, 21
- snpgdsEIGMIX, 6, 22, 23
- snpgdsErrMsg, 24
- snpgdsExampleFileName, 25
- SNPGDSFileClass, 8, 12, 19, 23, 26, 26, 28, 29, 33, 34, 37, 38, 41, 44, 45, 49, 50, 52, 54, 55, 58, 60, 61, 67, 69, 72, 73, 75, 78, 80, 81, 84, 85
- SNPGDSFileClass-class (SNPGDSFileClass), 26
- snpgdsFst, 26, 35, 55, 80
- snpgdsGDS2BED, 28, 31, 77
- snpgdsGDS2Eigen, 29
- snpgdsGDS2PED, 11, 28, 30, 30, 77
- snpgdsGEN2GDS, 31
- snpgdsGetGeno, 33
- snpgdsGRM, 34, 55
- snpgdsHCluster, 17, 18, 20, 35
- snpgdsHWE, 37
- snpgdsIBDKING, 38, 48
- snpgdsIBDMLE, 39, 40, 44, 46, 48, 63, 65
- snpgdsIBDMLELogLik, 42, 43, 46, 63, 65
- snpgdsIBDMoM, 39, 42, 44, 45, 48, 63, 65
- snpgdsIBDSelection, 47
- snpgdsIBS, 18, 36, 49, 51, 68
- snpgdsIBSNum, 50, 50
- snpgdsIndInb, 35, 51, 55
- snpgdsIndInbCoef, 53
- snpgdsIndivBeta, 54
- snpgdsLDMat, 55, 58, 59
- snpgdsLDpair, 56, 57, 59
- snpgdsLDpruning, 9, 56, 58, 58, 79
- snpgdsOpen, 12, 26, 60
- snpgdsOption, 10, 11, 61, 90, 91
- snpgdsPairIBD, 62, 65
- snpgdsPairIBDMLELogLik, 63, 64
- snpgdsPairScore, 66
- snpgdsPCA, 6, 23, 69, 72–76
- snpgdsPCACorr, 70, 72, 74, 76

snpGdsPCASampLoading, [70](#), [73](#), [73](#), [76](#)  
snpGdsPCASNPLoading, [70](#), [73](#), [74](#), [75](#)  
snpGdsPED2GDS, [11](#), [76](#)  
snpGdsSampMissRate, [77](#), [79](#), [85](#)  
snpGdsSelectSNP, [78](#)  
snpGdsSlidingWindow, [79](#)  
snpGdsSNPList, [81](#), [82–84](#)  
snpGdsSNPListClass, [13](#), [82](#), [82](#)  
snpGdsSNPListIntersect, [81](#), [82](#), [82](#)  
snpGdsSNPListStrand, [81](#), [83](#), [83](#), [84](#)  
snpGdsSNPRateFreq, [37](#), [78](#), [79](#), [84](#)  
snpGdsSummary, [85](#)  
snpGdsTranspose, [86](#)  
snpGdsVCF2GDS, [32](#), [87](#)  
snpGdsVCF2GDS\_R, [90](#), [91](#)  
SNPRelate (SNPRelate-package), [3](#)  
SNPRelate-package, [3](#)