Package 'GBScleanR'

June 28, 2022

Type Package

Title Error correction tool for noisy genotyping by sequencing (GBS) data

Version 1.0.0

Date 2022-03-24

Description GBScleanR is a package for quality check, filtering, and error correction of genotype data derived from next generation sequcener (NGS) based genotyping platforms. GBScleanR takes Variant Call Format (VCF) file as input. The main function of this package is `estGeno()` which estimates the true genotypes of samples from given read counts for genotype markers using a hidden Markov model with incorporating uneven observation ratio of allelic reads. This implementation gives robust genotype estimation even in noisy genotype data usually observed in Genotyping-By-Sequnencing (GBS) and similar methods, e.g. RADseq. The current implementation accepts genotype data of a diploid population at any generation of multi-parental cross, e.g. biparental F2 from inbred parents, biparental F2 from outbred parents, and 8-way recombinant inbred lines (8-way RILs) which can be refered to as MAGIC population.

License GPL-3 + file LICENSE

Encoding UTF-8

LinkingTo Rcpp, RcppParallel

SystemRequirements GNU make, C++11

Imports GWASTools, graphics, stats, utils, methods, gdsfmt, ggplot2, tidyr, SeqArray, Rcpp, RcppParallel, expm, Biobase

Suggests BiocStyle, testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

biocViews GeneticVariability, SNP, Genetics, HiddenMarkovModel, Sequencing, QualityControl

BugReports https://github.com/tomoyukif/GBScleanR/issues

URL https://github.com/tomoyukif/GBScleanR

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/GBScleanR

git_branch RELEASE_3_15

git_last_commit 9415bee

git_last_commit_date 2022-04-26

Date/Publication 2022-06-28

Author Tomoyuki Furuta [aut, cre] (<https://orcid.org/0000-0002-0869-6626>)

Maintainer Tomoyuki Furuta <f.tomoyuki@okayama-u.ac.jp>

R topics documented:

addScan *Add genotype data into the GDS file*

Description

Add genotype data into the GDS file

Usage

```
addScan(object, id, genotype, reads, ...)

## S4 method for signature 'GbsrGenotypeData'
addScan(object, id, genotype, reads)
```

Arguments

object	A GbsrGenotypeData object.
id	A character vector.
genotype	A numeric vector or matrix indicating genotypes at markers of given samples. The length or the number of columns should match with the number of markers recorded in the GDS file, which can be obtained via nsnp() with valid = FALSE.
reads	A numeric vector of matrix indicating read counts at markers of given samples. The length or the number of columns should match with twice the number of markers recorded in the GDS file, which can be obtained via nsnp() with valid = FALSE.
...	Unused.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

id <- "Dummy1"
genotype <- sample(c(0:3), nsnp(gds), replace = TRUE)
reads <- round(rexp(nsnp(gds) * 2, 1/5))
missing_pos <- which(genotype == 3)
reads[c(missing_pos, missing_pos + 1)] <- 0
ref_pos <- which(genotype == 2)
reads[ref_pos + 1] <- 0
alt_pos <- which(genotype == 0)
reads[alt_pos] <- 0

addScan(gds, id, genotype, reads)

# Close the connection to the GDS file
closeGDS(gds)
```

addScheme *#' Build a GbsrScheme object*

Description

GBScleanR uses breeding scheme information to set the expected number of cross overs in a chromosome which is a required parameter for the genotype error correction with the Hidden Markov model implemented in the `estGeno()` function. This function build the object storing type crosses performed at each generation of breeding and population sizes.

Usage

```
addScheme(object, crosstype, mating, pop_size, ...)

## S4 method for signature 'GbsrGenotypeData'
addScheme(object, crosstype, mating, pop_size)

## S4 method for signature 'GbsrScheme'
addScheme(object, crosstype, mating, pop_size)
```

Arguments

`object`	A GbsrGenotypeData object.
`crosstype`	A string to indicate the type of cross conducted with a given generation.
`mating`	An integer matrix to indicate mating combinations. The each element should match with member IDs of the last generation.
`pop_size`	An integer of the number of individuals in a given generation.
`...`	Unused.

Details

A scheme object is just a data.frame indicating a population size and a type of cross applied to each generation of the breeding process to generate the population which you are going to subject to the `estGeno()` function. The `crosstype` can take either of "selfing", "sibling", "pairing", and "random". When you set `crosstype = "random"`, you need to specify `pop_size` to indicate how many individuals were crossed in the random mating. You also need to specify a matrix indicating combinations of `mating`, in which each column shows a pair of member IDs indicating parental samples of the cross. Member IDs are serial numbers starts from 1 and automatically assigned by `initScheme()` and `addScheme()`. To check the member IDs, run `showScheme()`. Please see the examples section for more details of specifying a `mating` matrix. The created GbsrScheme object is set in the `scheme` slot of the GbsrGenotypeData object.

Value

A GbsrGenotypeData object storing a GbsrScheme object in the "scheme" slot.

See Also

addScheme() and showScheme()

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Biparental F2 population.
gds <- setParents(gds, parents = c("Founder1", "Founder2"))

# setParents gave member ID 1 and 2 to Founder1 and Founder2, respectively.
gds <- initScheme(gds, crosstype = "pair", mating = cbind(c(1:2)))

# Now the progenies of the cross above have member ID 3.
# If `crosstype = "selfing"` or `"sibling"`, you can omit a `mating` matrix.
gds <- addScheme(gds, crosstype = "self")

##############################################################################
# Now you can execute `estGeno()` which requires a [GbsrScheme] object.

# Close the connection to the GDS file
closeGDS(gds)
```

---

boxplotGBSR *Draw boxplots of specified statistics*

Description

Draw boxplots of specified statistics

Usage

```
boxplotGBSR(
  x,
  stats = "missing",
  target = c("snp", "scan"),
  q = 0.5,
  color = c(Marker = "darkblue", Sample = "darkblue"),
  fill = c(Marker = "skyblue", Sample = "skyblue")
)
```

Arguments

x	A GbsrGenotypeData object.
stats	A string to specify statistics to be drawn.

target	Either or both of "snp" and "scan", e.g. target = "snp" to draw a histogram only for SNPs.
q	An integer to specify a quantile calculated via calcReadStats().
color	A named vector "Marker" and "Sample" to specify border color of bins in the histograms.
fill	A named vector "Marker" and "Sample" to specify fill color of bins in the histograms.

Details

You can draw boxplots of several summary statistics of genotype counts and read counts per sample and per marker. The "stats" argument can take the following values:

- "missing""Proportion of missing genotype calls.",
- "het""Proportion of heterozygote calls.",
- "raf""Reference allele frequency.",
- "dp""Total read counts.",
- "ad_ref""Reference allele read counts.",
- "ad_alt""Alternative allele read counts.",
- "rrf""Reference allele read frequency.",
- "mean_ref""Mean of reference allele read counts.",
- "sd_ref""Standard deviation of reference allele read counts.",
- "qtile_ref""Quantile of reference allele read counts.",
- "mean_alt""Mean of alternative allele read counts.",
- "sd_alt""Standard deviation of alternative allele read counts.",
- "qtile_alt""Quantile of alternative allele read counts.",
- "mq""Mapping quality.",
- "fs""Phred-scaled p-value (strand bias)",
- "qd""Variant Quality by Depth",
- "sor""Symmetric Odds Ratio (strand bias)",
- "mqranksum""Alt vs. Ref read mapping qualities",
- "readposranksum""Alt vs. Ref read position bias",
- "baseqranksum""Alt Vs. Ref base qualities",

To draw boxplots for "missing", "het", "raf", you need to run countGenotype() first to obtain statistics. Similary, "dp", "ad_ref", "ad_alt", "rrf" requires values obtained via countRead(). calcReadStats() should be executed before drawing boxplots of "mean_ref", "sd_ref", "qtile_ref", "mean_alt", "sd_alt", and "qtile_alt". "mq", "fs", "qd", "sor", "mqranksum", "readposranksum", and "baseqranksum" only work with target = "snp", if your data contains those values supplied via SNP calling tools like GATK.

Value

A ggplot object.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gdata <- loadGDS(gds_fn)

# Summarize genotype count information to be used in `boxplotGBSR()`
gdata <- countGenotype(gdata)

boxplotGBSR(gdata, stats = "missing")

# Calculate means, standard deviations, quantile values of read counts
# to be used in `boxplotGBSR()`
gdata <- calcReadStats(gdata, q = 0.9)

# Draw boxplots of 90 percentile values of reference read counts and
# alternative read counts per SNP and per sample.
boxplotGBSR(gdata, stats = "qtile_ref", q = 0.9)

# Close the connection to the GDS file
closeGDS(gdata)
```

`calcReadStats` *Calculate mean, standard deviation, and quantile values of normalized read counts per sample and per marker.*

Description

This function first calculates normalized allele read counts by dividing allele read counts at each marker in each sample by the total allele read of the sample followed by multiplication by 10^6. In other words, it calculates reads per million (rpm). Then, the function calculates mean, standard deviation, quantile values of rpm per marker and per sample. The results will be stored in the SnpAnnotaionDataFrame slot and the ScanAnnotationDataFrame slot and obtained via getter functions, e.g. `getMeanReadRef()` and `getQtileReadAlt()`.

Usage

```
calcReadStats(object, target = "both", q = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
calcReadStats(object, target, q)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`q`	A numeric value [0-1] to indicate quantile to obtain.
`...`	Unused.

Details

Read count data can be obtained from the "annotation/format/AD/data" node or the "annotation/format/AD/filt.data" node of the GDS file with node = "raw" or node = "filt", respectively. The setCallFilter() function generate filtered read count data in the "annotation/format/AD/filt.data" node which can be accessed as mentioned above. The calculation of mean, standard deviation, and quantile values omits 0 in the read count data.

Value

A GbsrGenotypeData object with read statistics information.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Calculate means, standard deviations, quantiles of read counts
# per marker and per sample with or without standardization of
# the counts and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- calcReadStats(gds, q = 0.5)

# Get the means of reference allele read counts (rpm) per marker.
mean_reference_read_depth <- getMeanReadRef(gds, target = "snp")

# Get the 0.5 percentiles (medians) of the alternative allele
# read counts (rpm) per marker.
median_reference_read_depth <- getQtileReadAlt(gds,
                                               target = "snp",
                                               q = 0.5)
# Draw histograms of the means of reference allele read counts (rpm)
# per sample and marker.
histGBSR(gds, stats = "mean_ref")

# Draw histograms of the 0.5 percentiles (medians) of
# alternative allele read counts (rpm) per sample and marker.
histGBSR(gds, stats = "qtile_alt", q = 0.5)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

closeGDS *Close the connection to the GDS file*

Description

Close the connection to the GDS file linked to the given GbsrGenotypeData object.

Usage

```
closeGDS(object, verbose = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
closeGDS(object, verbose)
```

Arguments

object	A GbsrGenotypeData object.
verbose	if TRUE, show information.
...	Unused.

Value

NULL.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Close the connection to the GDS file
closeGDS(gds)
```

---

countGenotype *Count genotype calls and alleles per sample and per marker.*

Description

This function calculates several summary statistics of genotype calls and alleles per marker and per sample. Those values will be stored in the SnpAnnotaionDataFrame slot and the ScanAnnotationDataFrame slot and obtained via getter functions, e.g.s `getCountGenoRef()`, `getCountAlleleRef()`, and `getMAF()`.

Usage

```
countGenotype(object, target = "both", node = "raw", ...)

## S4 method for signature 'GbsrGenotypeData'
countGenotype(object, target, node)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
node	Either of "raw", "filt", and "cor". See details.
...	Unused.

Details

#' Genotype call data can be obtained from the "genotype" node, the "filt.genotype" node, or the "corrected.genotype" node of the GDS file with node = "raw", node = "filt", or node = "raw", respectively. The setCallFilter() function generate filtered genotype call data in the "filt.genotype" node which can be accessed as mentioned above. On the other hand, the "corrected.genotype" node can be generated via the estGeno() function.

Value

A GbsrGenotypeData object with genotype count information.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the genotype count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

# Get the proportion of missing genotype per sample.
sample_missing_rate <- getCountGenoMissing(gds,
                                           target = "scan",
                                           prop = TRUE)

# Get the minor allele frequency per marker.
marker_minor_allele_freq <- getMAF(gds, target = "snp")

# Draw histograms of the missing rate per sample and marker.
histGBSR(gds, stats = "missing")

# Close the connection to the GDS file.
closeGDS(gds)
```

countRead *Count reads per sample and per marker.*

Description

This function calculates several summary statistics of read counts per marker and per sample. Those values will be stored in the SnpAnnotaionDataFrame slot and the ScanAnnotationDataFrame slot and obtained via getter functions, e.g. `getCountReadRef()` and `getCountReadAlt()`.

Usage

```
countRead(object, target = "both", node = "raw", ...)

## S4 method for signature 'GbsrGenotypeData'
countRead(object, target, node)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
node	Either of "raw" and "filt". See details.
...	Unused.

Details

Read count data can be obtained from the "annotation/format/AD/data" node or the "annotation/format/AD/filt.data" node of the GDS file with `node = "raw"` or `node = "filt"`, respectively. The `setCallFilter()` function generate filtered read count data in the "annotation/format/AD/filt.data" node which can be accessed as mentioned above.

Value

A GbsrGenotypeData object with read count information.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the read count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countRead(gds)

# Get the total read counts per marker
read_depth_per_marker <- getCountRead(gds, target = "snp")
```

```
# Get the proportion of reference allele rads per marker.
reference_read_freq <- getCountReadRef(gds, target = "snp", prop = TRUE)

# Draw histgrams of reference allele read counts per sample and marker.
histGBSR(gds, stats = "ad_ref")

# Close the connection to the GDS file.
closeGDS(gds)
```

---

estGeno *Genotype estimation using a hiden Morkov model*

Description

Clean up genotype data by error correction based on genotype estimation using a hidden Markov model.

Usage

```
estGeno(
  object,
  chr,
  recomb_rate = 0.04,
  error_rate = 0.0025,
  call_threshold = 0.9,
  het_parent = FALSE,
  optim = TRUE,
  iter = 2,
  n_threads = NULL,
  ...
)

## S4 method for signature 'GbsrGenotypeData'
estGeno(
  object,
  chr,
  recomb_rate,
  error_rate,
  call_threshold,
  het_parent,
  optim,
  iter,
  n_threads
)
```

Arguments

object	A GbsrGenotypeData object.
chr	An integer vector of chromosome indices to be analyzed. All chromosomes will be analyzed if you left it default.
recomb_rate	A numeric value to indicate the expected recombination frequency per chromosome per megabase pairs.
error_rate	A numeric value of the expected sequence error rate.
call_threshold	A numeric value of the probability threshold to accept estimated genotype calls.
het_parent	A logical value to indicate whether parental samples are outbred or inbred. If FALSE, this function assume all true genotype of markers in parents are homozygotes.
optim	A logical value to specify whether to conduct parameter optimization for error correction.
iter	An integer value to specify the number of iterative parameter updates.
n_threads	An integer value to specify the number of threads used for the calculation. The default is n_threads = NULL and automatically set half the number of available threads on the computer.
...	Unused.

Value

A GbsrGenotypeData object in which the "estimated.haplotype", "corrected.genotype" and "parents.genotype" nodes were added.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Find the IDs of parental samples.
parents <- grep("Founder", getScanID(gds), value = TRUE)

# Set the parents and flip allele information
# if the reference sample (Founder1 in our case) has homozygous
# alternative genotype at some markers of which alleles will
# be swapped to make the reference sample have homozygous
# reference genotype.
gds <- setParents(gds, parents = parents, flip = TRUE)

# Initialize a scheme object stored in the slot of the GbsrGenotypeData.
# We chose `crosstype = "pair"` because two inbred founders were mated
# in our breeding scheme.
# We also need to specify the mating matrix which has two rows and
# one column with integers 1 and 2 indicating a sample (founder)
# with the memberID 1 and a sample (founder) with the memberID 2
# were mated.
```

```
gds <- initScheme(gds, crosstype = "pair", mating = cbind(c(1:2)))

# Add information of the next cross conducted in our scheme.
# We chose 'crosstype = "selfing"', which do not require a
# mating matrix.
gds <- addScheme(gds, crosstype = "selfing")

# Execute error correction by estimating genotype and haplotype of
# founders and offspring.
gds <- estGeno(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

gbsrGDS2CSV *Write a CSV file based on data in a GDS file*

Description

Write out a CSV file with raw, filtered, corrected genotype data or estimated haplotype data stored in a GDS file.

Usage

```
gbsrGDS2CSV(
  object,
  out_fn,
  node = "raw",
  incl_parents = TRUE,
  bp2cm = NULL,
  format = "",
  read = FALSE,
  ...
)

## S4 method for signature 'GbsrGenotypeData'
gbsrGDS2CSV(object, out_fn, node, incl_parents, bp2cm, format, read)
```

Arguments

object	A GbsrGenotypeData object.
out_fn	A string to specify the path to an output VCF file.
node	Either one of "raw", "filt", "cor", and "hap" to output raw genotype data, filtered genotype data, corrected genotype data, estimated haplotype data, respectively.
incl_parents	A logical value to specify whether parental samples should be included in an output VCF file or not.

bp2cm	A numeric value to convert positions in basepairs (bp) to centiMorgan (cm). The specified here is used to multiply position values. The default is NULL and then internally sets bp2cm = 4e-06 when format = "qtl. If not format = "qtl, 1 is set to bp2cm as default.
format	A string to indicate the output format. See details.
read	A logical value to indicate whether read counts should be output with genotype data or not. See details.
...	Unused.

Details

Create a CSV file at location specified by out_fn. The current implementation only changes the behavior when format = "qtl" to export the data in the r/qtl format that can be loaded using read.cross as format = "csvs with a phenotype data. Any other values are ignored and output a CSV file with the rows indicating chromosome ID and positions of markers followed by the rows indicating genotype or haplotype data of samples. If read = TRUE, the output of each genotype call would be in the form of GT:ADR,ADA where GT, ADR, and ADA represent genotype, referenece read count, and alternative read count, respectively. If format = "qtl", read = TRUE will be ignored.

Value

The path to the CSV file.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Create a CSV file with data from the GDS file
#  connected to the [GbsrGenotypeData] oobject.
out_fn <- tempfile("sample_out", fileext = ".csv")
gbsrGDS2CSV(gds, out_fn)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

gbsrGDS2VCF *Write a VCF file based on data in a GDS file*

Description

Write out a VCF file with raw, filtered, or corrected genotype data stored in a GDS file. The output VCF file contains the GT, AD, and DP fields.

Usage

```
gbsrGDS2VCF(object, out_fn, node = "raw", incl_parents = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
gbsrGDS2VCF(object, out_fn, node, incl_parents)
```

Arguments

object	A GbsrGenotypeData object.
out_fn	A string to specify the path to an output VCF file.
node	Either one of "raw", "filt", and "cor" to output raw genotype data, filtered genotype data, or corrected genotype data, respectively.
incl_parents	A logical value to specify whether parental samples should be included in an output VCF file or not.
...	Unused.

Details

Create a VCF file at location specified by out_fn. The connection to the GDS file of the input GbsrGenotypeData object will be automatically closed for internal file handling in this function. Please use `openGDS()` to open the connection again. If you use `loadGDS()`, summary statistics and filtering information will be discarded.

Value

The path to the VCF file.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Create a VCF file with data from the GDS file
#  connected to the [GbsrGenotypeData] oobject.
out_fn <- tempfile("sample_out", fileext = ".vcf.gz")
gbsrGDS2VCF(gds, out_fn)
```

GbsrGenotypeData-class

Class GbsrGenotypeData

Description

The GbsrGenotypeData class is the main class of GBScleanR and user work with this class object.

Details

The GbsrGenotypeData class has slots to store summarized genotype data, a GbsrScheme object, and the connection information to a GDS file which can be generated from a VCF file of your genotype data via gbsrVCF2GDS(). The GbsrGenotypeData class has four slots: data, snpAnnot, and scanAnnot and scheme. "Samples" or "individuals" subjected to genotyping are called "scan" here with following the way used in GWASTools. The slots snpAnnot and scanAnnot store a SnpAnnotationDataFrame object and a ScanAnnotationDataFrame object, respectively. The slot data stores a GdsGenotypeReader object. Those three classes are included from GWASTools. The slot scheme holds a GbsrScheme object. The function loadGDS() initialize those class objects internally.

Slots

data A GdsGenotypeReader object.

snpAnnot A SnpAnnotationDataFrame object.

scanAnnot A ScanAnnotationDataFrame object.

scheme A GbsrScheme object.

Examples

```
# `loadGDS()` initialize objects in the slots of a `GbsrGenotypeData`
# object internally.

# Load data in the GDS file and instantiate
# a `GbsrGenotypeData` object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gdata <- loadGDS(gds_fn)

# Close connection to the GDS file.
closeGDS(gdata)
```

GbsrScheme-class *Class* GbsrScheme

Description

GBScleanR uses breeding scheme information to set the expected number of cross overs in a chromosome which is a required parameter for the genotype error correction with the hidden Markov model implemented in the estGeno() function. This class stores those information including ID of parental samples, type crosses performed at each generation of breeding and population sizes of each generation. This class is not exported.

Slots

`crosstype` A vector of strings indicating the type of crossing done at each generation.

`pop_size` A vector of integers of the population size of each generation.

`mating` A list of matrices showing combinations member IDs of samples mated.

`parents` A vector of member IDs of parents.

`progenies` A vector of memeber IDs of progenies produced at each generation.

See Also

GbsrGenotypeData dnd `loadGDS()`.

Examples

```
# `loadGDS()` initialize a `GbsrScheme` object internally and
# attache it to the shceme slot of a [GbsrGenotypeData] object.

# Load data in the GDS file and instantiate
# a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gdata <- loadGDS(gds_fn)

# Print the information stored in the `GbsrScheme` object.
showScheme(gdata)

# Close the connection to the GDS file.
closeGDS(gdata)
```

---

gbsrVCF2GDS *Convert a VCF file to a GDS file*

Description

This function converts a variant call data in the VCF format. The current implementation only accepts biallelic single nucleotide polymorphisms. Please filter out variants which are insertions and deletions or multiallelic. You may use "bcftools" or "vcftools" for filtering.

Usage

```
gbsrVCF2GDS(vcf_fn, out_fn, force = FALSE, verbose = TRUE)
```

Arguments

`vcf_fn`	A string to indicate path to an input VCF file.
`out_fn`	A string to indicate path to an output GDS file.
`force`	A logical value to overwrite a GDS file even if the file specified in "out_fn" exists.
`verbose`	if TRUE, show information.

Details

gbsrVCF2GDS converts a VCF file to a GDS file. The data structure of the GDS file created via this functions is same with those created by `snpgdsVCF2GDS` of `SNPRelate`. Unlike the GDS files created by `SNPRelate` `GBScleanR`'s GDS files:

- "Include annotation information of variants recorded in the `INFO` filed of the input VCF."
- "Include allelic read count data (indicated as AD) recoreded in the `FORMAT` filed of the input VCF."

Value

The output GDS file path.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a `GbsrGenotypeData` object.
gdata <- loadGDS(gds_fn)

# Close the connection to the GDS file.
closeGDS(gdata)
```

`getAlleleA` *Obtain reference allele information of each SNP marker*

Description

This function returns reference alleles, either of A, T, G, and C, of SNP markers.

Usage

```
getAlleleA(object, valid = TRUE, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
getAlleleA(object, valid, chr)
```

Arguments

`object`	A GbsrGenotypeData object.
`valid`	A logical value. See details.
`chr`	A index to spefcify chromosome to get information.
`...`	Unused.

Details

If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A character vector indicating the reference alleles.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getAlleleA(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getAlleleB` *Obtain alternative allele information of each SNP marker*

Description

This function returns alternative alleles, either of A, T, G, and C, of SNP markers.

Usage

```
getAlleleB(object, valid = TRUE, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
getAlleleB(object, valid, chr)
```

Arguments

`object`	A GbsrGenotypeData object.
`valid`	A logical value. See details.
`chr`	A index to spefcify chromosome to get information.
`...`	Unused.

Details

If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A character vector indicating the alternative alleles.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getAlleleB(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getChromosome *Obtain chromosome information of each SNP marker*

Description

This function returns indexes or names of chromsomes of each SNP or just a set of unique chromosome names.

Usage

```
getChromosome(object, valid = TRUE, levels = FALSE, name = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getChromosome(object, valid, levels, name)
```

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
levels	A logical value. See details.
name	A logical value. See details.
...	Unused.

Details

A GDS file created via GBScleanR stores chromosome names as sequential integers from 1 to N, where N is the number of chromosomes. This function returns those indexes as default. If you need actual names of the chromosomes, set name = TRUE. levels = TRUE gives you only unique chromosome names with length N. If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A vector of integers or strings indicating chromosome IDs.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getChromosome(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getCountAlleleAlt *Obtain total alternative allele counts per SNP or per scan (sample)*

Description

Obtain total alternative allele counts per SNP or per scan (sample)

Usage

```
getCountAlleleAlt(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountAlleleAlt(object, target, valid, prop)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
prop	A logical value whether to return values as proportions of total alternative allele counts to total non missing allele counts or not.
...	Unused.

Details

You need to execute countGenotype() to calculate sumaary statisticsto be obtained via this function. If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A numeric vector of (proportion of) alternative alleles per marker.

Examples

```
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)
gds <- countGenotype(gds)
getCountAlleleAlt(gds)
closeGDS(gds) # Close the connection to the GDS file
```

`getCountAlleleMissing` *Obtain total missing allele counts per SNP or per scan (sample)*

Description

Obtain total missing allele counts per SNP or per scan (sample)

Usage

```
getCountAlleleMissing(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountAlleleMissing(object, target, valid, prop)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`valid`	A logical value. See details.
`prop`	A logical value whether to return values as proportions of total missing allele counts to the total allele number or not.
`...`	Unused.

Details

You need to execute `countGenotype()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of (proportion of) missing alleles per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the genotype count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

getCountAlleleMissing(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getCountAlleleRef *Obtain total reference allele counts per SNP or per scan (sample)*

Description

Obtain total reference allele counts per SNP or per scan (sample)

Usage

```
getCountAlleleRef(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountAlleleRef(object, target, valid, prop)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
prop	A logical value whether to return values as proportions of total reference allele counts to total non missing allele counts or not.
...	Unused.

Details

You need to execute `countGenotype()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of (proportion of) reference alleles per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the genotype count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

getCountAlleleRef(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getCountGenoAlt` *Obtain total alternative genotype counts per SNP or per scan (sample)*

Description

Obtain total alternative genotype counts per SNP or per scan (sample)

Usage

```
getCountGenoAlt(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountGenoAlt(object, target, valid, prop)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`valid`	A logical value. See details.
`prop`	A logical value whether to return values as proportions of total alternative genotype counts to total non missing genotype counts or not.
`...`	Unused.

Details

You need to execute `countGenotype()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of (proportion of) homozygous alternative genotype calls per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the genotype count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

getCountGenoAlt(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

---

getCountGenoHet *Obtain total heterozygote counts per SNP or per scan (sample)*

Description

Obtain total heterozygote counts per SNP or per scan (sample)

Usage

```
getCountGenoHet(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountGenoHet(object, target, valid, prop)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
prop	A logical value whether to return values as proportions of total heterozygote counts to total non missing genotype counts or not.
...	Unused.

Details

You need to execute `countGenotype()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of (proportion of) heterozugous genotype calls per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the genotype count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

getCountGenoHet(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getCountGenoMissing *Obtain total missing genotype counts per SNP or per scan (sample)*

Description

Obtain total missing genotype counts per SNP or per scan (sample)

Usage

```
getCountGenoMissing(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountGenoMissing(object, target, valid, prop)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
prop	A logical value whether to return values as proportions of total missing genotype counts to the total genotype calls or not.
...	Unused.

Details

You need to execute `countGenotype()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of (proportion of) missing genotype calls per marker.

Examples

```
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)
gds <- countGenotype(gds)
getCountGenoMissing(gds)
closeGDS(gds) # Close the connection to the GDS file
```

`getCountGenoRef` *Obtain total reference genotype counts per SNP or per scan (sample)*

Description

Obtain total reference genotype counts per SNP or per scan (sample)

Usage

```
getCountGenoRef(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountGenoRef(object, target, valid, prop)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`valid`	A logical value. See details.
`prop`	A logical value whether to return values as proportions of total reference genotype counts to total non missing genotype counts or not.
`...`	Unused.

Details

You need to execute `countGenotype()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of (proportion of) homozygous reference genotype calls per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the genotype count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

getCountGenoRef(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getCountRead` *Obtain total read counts per SNP or per scan (sample)*

Description

Obtain total read counts per SNP or per scan (sample)

Usage

```
getCountRead(object, target = "snp", valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountRead(object, target, valid)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`valid`	A logical value. See details.
`...`	Unused.

Details

You need to execute `countRead()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A integer vector of total read counts (reference allele reads + alternative allele reads) per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getCountRead(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getCountReadAlt` *Obtain total alternative read counts per SNP or per scan (sample)*

Description

Obtain total alternative read counts per SNP or per scan (sample)

Usage

```
getCountReadAlt(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountReadAlt(object, target, valid, prop)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`valid`	A logical value. See details.
`prop`	A logical value whether to return values as proportions of total alternative read counts in total read counts per SNP or not.
`...`	Unused.

Details

You need to execute `countRead()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of (proportion of) alternative allele read counts per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getCountReadAlt(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getCountReadRef` *Obtain total reference read counts per SNP or per scan (sample)*

Description

Obtain total reference read counts per SNP or per scan (sample)

Usage

```
getCountReadRef(object, target = "snp", valid = TRUE, prop = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getCountReadRef(object, target, valid, prop)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`valid`	A logical value. See details.
`prop`	A logical value whether to return values as proportions of total reference read counts in total read counts per SNP or not.
`...`	Unused.

Details

You need to execute `countRead()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of (proportion of) reference read counts per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getCountReadRef(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getFlipped` *Get a logical vector indicating flipped SNP markers.*

Description

Get a logical vector indicating flipped SNP markers.

Usage

```
getFlipped(object, valid = TRUE, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
getFlipped(object, valid, chr)
```

Arguments

`object`	A GbsrGenotypeData object.
`valid`	A logical value to specify flip alleles only of valid markers. see also`setSnpFilter()`.
`chr`	A index to spefcify chromosome to get information.
`...`	Unused.

Details

Flipped markers are markers where the alleles expected as reference allele are called as alternative allele. If you specify two parents in the `parents` argument of `setParents()` with `flip = TRUE`, `bi = TRUE`, and `homo = TRUE`, the alleles found in the parent specified as the first element to the `parents` argument are supposed as reference alleles of the markers. If the "expected" reference alleles are not actually called as reference alleles but alternative alleles in the given data. `setParents()` will automatically labels those markers "flipped". The SnpAnnotatoinDataFrame slot sores this information and accessible via `getFlipped()` which gives you a logical vector indicating which markers are labeled as flipped `TRUE` or not flipped `FALSE`. `hasFlipped()` just tells you whether the SnpAnnotatoinDataFrame slot has the information of flipped markers or not.

Value

A logical vector indicating which markers of alleles were flipped.

See Also

setParents() and hasFlipped().

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Find the IDs of parental samples.
parents <- grep("Founder", getScanID(gds), value = TRUE)

# Set the parents and flip allele information
# if the reference sample (Founder1 in our case) has homozygous
# alternative genotype at some markers of which alleles will
# be swapped to make the reference sample have homozygous
# reference genotype.
gds <- setParents(gds, parents = parents, flip = TRUE)

getFlipped(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getGenotype *Get genotype call data.*

Description

Genotype calls are retrieved from the GDS file linked to the given GbsrGenotypeData object.

Usage

```
getGenotype(
  object,
  valid = TRUE,
  chr = NULL,
  node = "raw",
  parents = FALSE,
  ...
)

## S4 method for signature 'GbsrGenotypeData'
getGenotype(object, valid, chr, node, parents)
```

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
chr	A integer vector of indexes indicating chromosomes to get read count data.
node	Either of "raw", "filt", and "cor. See details.
parents	A logical value or "only" to include data for parents or to get data only for parents.
...	Unused.

Details

Genotype call data can be obtained from the "genotype" node, the "filt.genotype" node, or the "corrected.genotype" node of the GDS file with `node = "raw"`, `node = "filt"`, or `node = "raw"`, respectively. If `node = "parents`, the data in the "parents.genotype" node will be returned. The "parents.genotype" node stores phased parental genotypes estimated by the `estGeno()` function. The `setCallFilter()` function generate filtered genotype call data in the "filt.genotype" node which can be accessed as mentioned above. On the other hand, the "corrected.genotype" node can be generated via the `estGeno()` function. If `valid = TRUE`, read counts for only valid marker and valid samples will be obtained.

Value

An integer matirix of genotype data which is represented by the number of reference alleles at each marker of each sample.

See Also

`setCallFilter()` and `estGeno()`

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

geno <- getGenotype(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getHaplotype` *Get haplotype call data.*

Description

Haplotype calls are retrieved from the GDS file linked to the given GbsrGenotypeData object.

Usage

```
getHaplotype(object, valid = TRUE, chr = NULL, parents = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getHaplotype(object, chr, parents)
```

Arguments

`object`	A GbsrGenotypeData object.
`valid`	A logical value. See details.
`chr`	A integer vector of indexes indicating chromosomes to get read count data.
`parents`	A logical value or "only" to include data for parents or to get data only for parents.
`...`	Unused.

Details

Haplotype call data can be obtained from the "estimated.haplotype" node of the GDS file which can be generated via the `estGeno()` function. Thus, this function is valid only after having executed `estGeno()`. If `valid = TRUE`, read counts for only valid marker and valid samples will be obtained.

Value

An integer array of haplotype data. The array have 2 x M x N dimensions, where M is the number of markers and N is the number of samples. Each integer values represent the origin of the haplotype. For example, in the population with two inbred founders, values take either 1 or 2 indicating the hapotype descent from founder 1 and 2. If two outbred founders, values take 1, 2, 3, or 4 indicating the first and second haplotype in founder 1 and the first and second haplotype in founder 2.

See Also

`estGeno()`

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Find the IDs of parental samples.
parents <- grep("Founder", getScanID(gds), value = TRUE)

# Set the parents and flip allele information
# if the reference sample (Founder1 in our case) has homozygous
# alternative genotype at some markers of which alleles will
# be swapped to make the reference sample have homozygous
# reference genotype.
gds <- setParents(gds, parents = parents, flip = TRUE)

# Initialize a scheme object stored in the slot of the GbsrGenotypeData.
# We chose `crosstype = "pair"` because two inbred founders were mated
# in our breeding scheme.
# We also need to specify the mating matrix which has two rows and
# one column with integers 1 and 2 indicating a sample (founder)
# with the memberID 1 and a sample (founder) with the memberID 2
# were mated.
gds <- initScheme(gds, crosstype = "pair", mating = cbind(c(1:2)))

# Add information of the next cross conducted in our scheme.
# We chose 'crosstype = "selfing"', which do not require a
# mating matrix.
gds <- addScheme(gds, crosstype = "selfing")

# Execute error correction by estimating genotype and haplotype of
# founders and offspring.
gds <- estGeno(gds)

hap <- getHaplotype(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getInfo` *Obtain information stored in the "annotation/info" node*

Description

The "annotation/info" node stores annotation infromation of markers obtained via SNP calling tools like bcftools and GATK.

Usage

```
getInfo(object, var, valid = TRUE, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
getInfo(object, var, valid, chr)
```

Arguments

object	A GbsrGenotypeData object.
var	A string to indicate which annotation info should be retrieved.
valid	A logical value. See details.
chr	A index to spefcify chromosome to get information.
...	Unused.

Details

If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A numeric vector of data stored in INFO node of the GDS file.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Get mapping qualities (MQ) of markers.
mq <- getInfo(gds, "MQ")

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getMAC *Obtain minor allele counts per SNP or per scan (sample)*

Description

Obtain minor allele counts per SNP or per scan (sample)

Usage

```
getMAC(object, target = "snp", valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getMAC(object, target, valid)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
...	Unused.

Details

You need to execute countGenotype() to calculate sumaary statisticsto be obtained via this function. If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A numeric vector of the minor allele counts per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the genotype count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

getMAC(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getMAF *Obtain minor allele frequencies per SNP or per scan (sample)*

Description

Obtain minor allele frequencies per SNP or per scan (sample)

Usage

```
getMAF(object, target = "snp", valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getMAF(object, target, valid)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
...	Unused.

Details

You need to execute countGenotype() to calculate sumaary statisticsto be obtained via this function. If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A numeric vector of the minor allele frequencies per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the genotype count information and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

getMAF(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

---

getMeanReadAlt *Obtain mean values of total alternative read counts per SNP or per scan (sample)*

Description

Obtain mean values of total alternative read counts per SNP or per scan (sample)

Usage

```
getMeanReadAlt(object, target = "snp", valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getMeanReadAlt(object, target, valid)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
...	Unused.

Details

You need to execute calcReadStats() to calculate sumaary statisticsto be obtained via this function. If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A numeric vector of the mean values of alternative allele reads per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Calculate means, standard deviations, quantiles of read counts
# per marker and per sample with or without standardization of
# the counts and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- calcReadStats(gds)

getMeanReadAlt(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getMeanReadRef *Obtain mean values of total reference read counts per SNP or per scan (sample)*

Description

Obtain mean values of total reference read counts per SNP or per scan (sample)

Usage

```
getMeanReadRef(object, target = "snp", valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getMeanReadRef(object, target, valid)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
...	Unused.

Details

You need to execute calcReadStats() to calculate sumaary statisticsto be obtained via this function. If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A numeric vector of the mean values of reference allele reads per marker.

Examples

```
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)
gds <- calcReadStats(gds)
getMeanReadRef(gds)
closeGDS(gds) # Close the connection to the GDS file
```

---

getParents *Get parental sample information*

Description

This function returns scan IDs, member IDs and indexes of parental samples set via setParents(). Scan IDs are IDs given by user or obtained from the original VCF file. Member IDs are serial numbers assigned by setParents().

Usage

```
getParents(object, bool = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getParents(object, bool)
```

Arguments

object	A GbsrGenotypeData object.
bool	If TRUE, the function returns a logical vector indicating which scans (samples) have been set as parents.
...	Unused.

Value

A data frame of parents information indicating scanIDs, memberIDs and indexes of parental lines assigned via setParents().

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Find the IDs of parental samples.
parents <- grep("Founder", getScanID(gds), value = TRUE)

# Set the parents.
gds <- setParents(gds, parents = parents, flip = TRUE)

# Get the information of parents.
getParents(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getPloidy *Obtain ploidy information of each SNP marker*

Description

This function returns ploidy of each SNP marker. The ploidy of all the markers in a dataset is a same value and the current implementation of GBScleanR only works with data having ploidy = 2 for all markers.

Usage

```
getPloidy(object, valid = TRUE, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
getPloidy(object, valid, chr)
```

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
chr	A index to spefcify chromosome to get information.
...	Unused.

Details

If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A integer vector indicating the ploidy of each marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getPloidy(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getPosition *Obtain physical position information of each SNP marker*

Description

This function returns physical positions of SNP markers.

Usage

```
getPosition(object, valid = TRUE, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
getPosition(object, valid, chr)
```

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
chr	A index to spefcify chromosome to get information.
...	Unused.

Details

If valid = TRUE, the chromosome information of markers which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. getValidSnp() tells you which samples are valid.

Value

A integer vetor indicating the physical positions of markers.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getPosition(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getQtileReadAlt` *Obtain quantile values of total alternative read counts per SNP or per scan (sample)*

Description

Obtain quantile values of total alternative read counts per SNP or per scan (sample)

Usage

```
getQtileReadAlt(object, target = "snp", q = 0.5, valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getQtileReadAlt(object, target, q, valid)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`q`	A numeric value [0-1] to indicate quantile to obtain.
`valid`	A logical value. See details.
`...`	Unused.

Details

You need to execute `calcReadStats()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of the quantile values of alternative allele reads per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Calculate means, standard deviations, quantiles of read counts
# per marker and per sample with or without standardization of
# the counts and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- calcReadStats(gds)

getQtileReadAlt(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getQtileReadRef` *Obtain quantile values of total reference read counts per SNP or per scan (sample)*

Description

Obtain quantile values of total reference read counts per SNP or per scan (sample)

Usage

```
getQtileReadRef(object, target = "snp", q = 0.5, valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getQtileReadRef(object, target, q, valid)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`q`	A numeric value [0-1] to indicate quantile to obtain.
`valid`	A logical value. See details.
`...`	Unused.

Details

You need to execute `calcReadStats()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of the quantile values of alternative allele reads per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Calculate means, standard deviations, quantiles of read counts
# per marker and per sample with or without standardization of
# the counts and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- calcReadStats(gds)

getQtileReadRef(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getRead *Get read count data.*

Description

Read counts for reference allele and alternative allele are retrieved from the GDS file linked to the given GbsrGenotypeData object.

Usage

```
getRead(object, valid = TRUE, chr = NULL, node = "raw", parents = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getRead(object, valid, chr, node, parents)
```

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
chr	A integer vector of indexes indicating chromosomes to get read count data.
node	Either of "raw" and "filt". See details.
parents	A logical value or "only" to include data for parents or to get data only for parents.
...	Unused.

Details

Read count data can be obtained from the "annotation/format/AD/data" node, the "annotation/format/AD/norm" node or the "annotation/format/AD/filt.data" node of the GDS file with `node = "raw"`, `node = "norm"`, or `node = "filt"`, respectively. The `calcReadStats()` function generate normalized read count data in the "annotation/format/AD/norm" node, while `setCallFilter()` function generate filtered read count data in the "annotation/format/AD/filt.data" node which can be accessed as mentioned above. If `valid = TRUE`, read counts for only valid marker and valid samples will be obtained.

Value

A named list with two elements "ref" and "alt" storing a matrix of reference allele read counts and a matrix of alternative read counts for all markers in all samples.

See Also

`setCallFilter()`

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

read <- getRead(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getScanID` *Obtain scan (sample) ID*

Description

This function returns scan (sample) ID.

Usage

```
getScanID(object, valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getScanID(object, valid)
```

Arguments

`object`	A GbsrGenotypeData object.
`valid`	A logical value. See details.
`...`	Unused.

Details

If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A character vector of scan(sample) IDs.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getScanID(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getSDReadAlt *Obtain standard deviations of total alternative read counts per SNP or per scan (sample)*

Description

Obtain standard deviations of total alternative read counts per SNP or per scan (sample)

Usage

```
getSDReadAlt(object, target = "snp", valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getSDReadAlt(object, target, valid)
```

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "snp" and "scan".
`valid`	A logical value. See details.
`...`	Unused.

Details

You need to execute `calcReadStats()` to calculate sumaary statisticsto be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of the standard deviations of alternative allele reads per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Calculate means, standard deviations, quantiles of read counts
# per marker and per sample with or without standardization of
# the counts and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- calcReadStats(gds)

getSDReadAlt(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getSDReadRef *Obtain standard deviations of total reference read counts per SNP or per scan (sample)*

Description

Obtain standard deviations of total reference read counts per SNP or per scan (sample)

Usage

```
getSDReadRef(object, target = "snp", valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
getSDReadRef(object, target, valid)
```

Arguments

object	A GbsrGenotypeData object.
target	Either of "snp" and "scan".
valid	A logical value. See details.
...	Unused.

Details

You need to execute `calcReadStats()` to calculate summary statistics to be obtained via this function. If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A numeric vector of the standard deviations of reference allele reads per marker.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Calculate means, standard deviations, quantiles of read counts
# per marker and per sample with or without standardization of
# the counts and store them in the
# [SnpAnnotationDataFrame] and [ScanAnnotationDataFrame] objects
# linked at the slots of the [GbsrGenotypeData] object.
gds <- calcReadStats(gds)

getSDReadRef(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

getSnpID *Obtain SNP ID*

Description

This function returns SNP ID of SNP markers.

Usage

```
getSnpID(object, valid = TRUE, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
getSnpID(object, valid, chr)
```

Arguments

`object`	A GbsrGenotypeData object.
`valid`	A logical value. See details.
`chr`	A index to spefcify chromosome to get information.
`...`	Unused.

Details

If `valid = TRUE`, the chromosome information of markers which are labeled `TRUE` in the ScanAnnotationDataFrame slot will be returned. `getValidSnp()` tells you which samples are valid.

Value

A character vector of SNP IDs.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getSnpID(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`getValidScan` *Return a logical vector indicating which are valid scans (samples).*

Description

Return a logical vector indicating which are valid scans (samples).

Usage

```
getValidScan(object, parents = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
getValidScan(object, parents)
```

Arguments

`object`	A GbsrGenotypeData object.
`parents`	A logical value to indicate to set FALSE or TRUE to parental samples. If you specify `parents = "only"`, this function returns a logical vector indicating TRUE for only parental samples.
`...`	Unused.

Value

A logical vector of the same length with the number of total samples.

See Also

setValidScan()

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getValidScan(gds)

# Close the connection the GDS file.
closeGDS(gds)
```

---

getValidSnp *Return a logical vector indicating which are valid SNP markers.*

Description

Return a logical vector indicating which are valid SNP markers.

Usage

```
getValidSnp(object, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
getValidSnp(object, chr)
```

Arguments

object	A GbsrGenotypeData object.
chr	A index to spefcify chromosome to get information.
...	Unused.

Value

A logical vector of the same length with the number of total SNP markers

See Also

setValidSnp()

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

getValidSnp(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

`hasFlipped` *Get a logical value indicating flipped SNP markers whether information exists.*

Description

Get a logical value indicating flipped SNP markers whether information exists.

Usage

```
hasFlipped(object, ...)

## S4 method for signature 'GbsrGenotypeData'
hasFlipped(object)
```

Arguments

`object`	A GbsrGenotypeData object.
`...`	Unused.

Details

Flipped markers are markers where the alleles expected as reference allele are called as alternative allele. If you specify two parents in the `parents` argument of `setParents()` with `flip = TRUE`, `bi = TRUE`, and `homo = TRUE`, the alleles found in the parent specified as the first element to the `parents` argument are supposed as reference alleles of the markers. If the "expected" reference alleles are not actually called as reference alleles but alternative alleles in the given data. `setParents()` will automatically labels those markers "flipped". The SnpAnnotatoinDataFrame slot sores this information and accessible via `getFlipped()` which gives you a logical vector indicating which markers are labeled as flipped `TRUE` or not flipped `FALSE`. `hasFlipped()` just tells you whether the SnpAnnotatoinDataFrame slot has the information of flipped markers or not.

Value

A logical value to indicate the GbsrGenotypeData contains markers at which alleles were flipped.

See Also

setParents() and getFlipped().

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Find the IDs of parental samples.
parents <- grep("Founder", getScanID(gds), value = TRUE)

# Set the parents and flip allele information
# if the reference sample (Founder1 in our case) has homozygous
# alternative genotype at some markers of which alleles will
# be swapped to make the reference sample have homozygous
# reference genotype.
gds <- setParents(gds, parents = parents, flip = TRUE)

hasFlipped(gds)

# Close the connection to the GDS file.
closeGDS(gds) # Close the connection to the GDS file
```

---

histGBSR *Draw histograms of specified statistics*

Description

Draw histograms of specified statistics

Usage

```
histGBSR(
  x,
  stats = c("dp", "missing", "het"),
  target = c("snp", "scan"),
  q = 0.5,
  binwidth = NULL,
  color = c(Marker = "darkblue", Sample = "darkblue"),
  fill = c(Marker = "skyblue", Sample = "skyblue")
)
```

Arguments

x	A GbsrGenotypeData object.
stats	A string to specify statistics to be drawn.
target	Either or both of "snp" and "scan", e.g. target = "snp" to draw a histogram only for SNPs.
q	An integer to specify a quantile calculated via calcReadStats().
binwidth	An integer to specify bin width of the histogram. This value is passed to the ggplot function.
color	A named vector "Marker" and "Sample" to specify border color of bins in the histograms.
fill	A named vector "Marker" and "Sample" to specify fill color of bins in the histograms.

Details

You can draw histograms of several summary statistics of genotype counts and read counts per sample and per marker. The "stats" argument can take the following values:

- "missing""Proportion of missing genotype calls.",
- "het""Proportion of heterozygote calls.",
- "raf""Reference allele frequency.",
- "dp""Total read counts.",
- "ad_ref""Reference allele read counts.",
- "ad_alt""Alternative allele read counts.",
- "rrf""Reference allele read frequency.",
- "mean_ref""Mean of reference allele read counts.",
- "sd_ref""Standard deviation of reference allele read counts.",
- "qtile_ref""Quantile of reference allele read counts.",
- "mean_alt""Mean of alternative allele read counts.",
- "sd_alt""Standard deviation of alternative allele read counts.",
- "qtile_alt""Quantile of alternative allele read counts.",
- "mq""Mapping quality.",
- "fs""Phred-scaled p-value (strand bias)",
- "qd""Variant Quality by Depth",
- "sor""Symmetric Odds Ratio (strand bias)",
- "mqranksum""Alt vs. Ref read mapping qualities",
- "readposranksum""Alt vs. Ref read position bias",
- "baseqranksum""Alt Vs. Ref base qualities",

To draw histograms for "missing", "het", "raf", you need to run countGenotype() first to obtain statistics. Similary, "dp", "ad_ref", "ad_alt", "rrf" requires values obtained via countRead(). calcReadStats() should be executed before drawing histograms of "mean_ref", "sd_ref", "qtile_ref", "mean_alt", "sd_alt", and "qtile_alt". "mq", "fs", "qd", "sor", "mqranksum", "readposranksum", and "baseqranksum" only work with target = "snp", if your data contains those values supplied via SNP calling tools like GATK.

Value

A ggplot object.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gdata <- loadGDS(gds_fn)

# Summarize genotype count information to be used in `histGBSR()`
gdata <- countGenotype(gdata)

# Draw histograms of missing rate, heterozygosity, and reference
# allele frequency per SNP and per sample.
histGBSR(gdata, stats = "missing")


# Calculate means, standard deviations, quantile values of read counts
# to be used in `histGBSR()`
gdata <- calcReadStats(gdata, q = 0.9)

# Draw histograms of 90 percentile values of reference read counts
# and alternative read counts per SNP and per sample.
histGBSR(gdata, stats = "qtile_ref", q = 0.9)

# Close the connection to the GDS file
closeGDS(gdata)
```

---

initScheme *Build a GbsrScheme object*

Description

GBScleanR uses breeding scheme information to set the expected number of cross overs in a chromosome which is a required parameter for the genotype error correction with the hidden Markov model implemented in the `estGeno()` function. This function build the object storing type crosses performed at each generation of breeding and population sizes.

Usage

```
initScheme(object, crosstype, mating, ...)

## S4 method for signature 'GbsrGenotypeData'
initScheme(object, crosstype, mating)

## S4 method for signature 'GbsrScheme'
initScheme(object, crosstype, mating, parents)
```

Arguments

object	A GbsrGenotypeData object.
crosstype	A string to indicate the type of cross conducted with a given generation.
mating	An integer matrix to indicate mating combinations. The each element should match with IDs of parental samples which are 1 to N. see Details.
...	Unused.
parents	Indices of parental lines.

Details

A GbsrScheme object stores information of a population size, mating combinations and a type of cross applied to each generation of the breeding process to generate the population which you are going to subject to the `estGeno()` function. The first generation should be parents of the population. It is supposed that `setParents()` has been already executed and parents are labeled in the GbsrGenotypeData object. The number of parents are automatically recognized. The "crosstype" of the first generation can be "pairing" or "random" with `pop_size = N`, where N is the number of parents. You need to specify a matrix indicating combinations of `mating`, in which each column shows a pair of parental samples. For example, if you have only two parents, the `mating` matrix is `mating = cbind(c(1:2))`. The indices used in the matrix should match with the IDs labeled to parental samples by `setParents()`. The created GbsrScheme object is set in the `scheme` slot of the GbsrGenotypeData object.

Value

A GbsrGenotypeData object storing a GbsrScheme object in the "scheme" slot.

See Also

`addScheme()` and `showScheme()`

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Biparental F2 population.
gds <- setParents(gds, parents = c("Founder1", "Founder2"))

# setParents gave member ID 1 and 2 to Founder1 and Founder2, respectively.
gds <- initScheme(gds, crosstype = "pair", mating = cbind(c(1:2)))

# Now the progenies of the cross above have member ID 3.
# If `crosstype = "selfing"` or `"sibling"`, you can omit a `mating` matrix.
gds <- addScheme(gds, crosstype = "self")

# Now you can execute `estGeno()` which requires a [GbsrScheme] object.

# Close the connection to the GDS file
```

```
closeGDS(gds)
```

`isOpenGDS` *Check if a GDS file has been opened or not.*

Description

Check if a GDS file has been opened or not.

Usage

```
isOpenGDS(object, ...)

## S4 method for signature 'GbsrGenotypeData'
isOpenGDS(object)
```

Arguments

`object`	A GbsrGenotypeData object.
`...`	Unused.

Value

`TRUE` if the GDS file linked to the input GbsrGenotypeData object has been opened, while `FALSE` if closed.

Examples

```
# Use a GDS file of example data.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")

# Instantiation of [GbsrGenotypeData]
gds <- loadGDS(gds_fn)

# Check connection to the GDS file
isOpenGDS(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

loadGDS *Load a GDS file and construct a* GbsrGenotypeData *object.*

Description

Load data stored in an input GDS file to R environment and create a GbsrGenotypeData instance. GBScleanR handles only one class GbsrGenotypeData and conducts all data manipulation via class methods for it.

Usage

```
loadGDS(x, verbose = TRUE)
```

Arguments

x	A string of the path to an input GDS file or a GbsrGenotypeData object to reload.
verbose	if TRUE, show information.

Value

A GbsrGenotypeData object.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a `GbsrGenotypeData` object.
gdata <- loadGDS(gds_fn)

# Reload data from the GDS file.
gdata <- loadGDS(gdata)

# Close the connection to the GDS file.
closeGDS(gdata)
```

loadScanAnnot *Load the stored ScanAnnotationDataFrame information*

Description

All the data stored in the ScanAnnotationDataFrame slot of the GbsrGenotypeData object can be saved in the GDS file linked to the given GbsrGenotypeData object via `saveScanAnnot()`. You can load the saved data using this function.

Usage

```
loadScanAnnot(object, ...)

## S4 method for signature 'GbsrGenotypeData'
loadScanAnnot(object)
```

Arguments

`object`	A GbsrGenotypeData object
`...`	Unused.

Value

A GbsrGenotypeData object.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Save data stored in a [ScanAnnotationDataFrame] object
# linked in the slot of the `GbsrGenotypeData.`
gds <- saveScanAnnot(gds)

# Close the connection to the GDS file
closeGDS(gds)

# Open the connection to the GDS file again.
gds <- loadGDS(gds)

# Load the saved [ScanAnnotationDataFrame] object
gds <- loadScanAnnot(gds)

# Close the connection to the GDS file
```

```
closeGDS(gds)
```

---

loadSnpAnnot *Load the stored SnpAnnotationDataFrame information*

Description

All the data stored in the SnpAnnotationnDataFrame slot of the GbsrGenotypeData object can be saved in the GDS file linked to the given GbsrGenotypeData object via saveSnpAnnot(). You can load the saved data using this function.

Usage

```
loadSnpAnnot(object, ...)

## S4 method for signature 'GbsrGenotypeData'
loadSnpAnnot(object)
```

Arguments

object	A GbsrGenotypeData object
...	Unused.

Value

A GbsrGenotypeData object.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Save data stored in a [ScanAnnotationDataFrame] object
# linked in the slot of the `GbsrGenotypeData.`
gds <- saveSnpAnnot(gds)

# Close the connection to the GDS file
closeGDS(gds)

# Open the connection to the GDS file again.
gds <- loadGDS(gds)

# Load the saved [SnpAnnotationDataFrame] object
```

```
gds <- loadSnpAnnot(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

---

nscan *Return the number of scans (samples).*

Description

This function returns the number of samples recorded in the GDS file connected to the given GbsrGenotypeData object.

Usage

```
nscan(object, valid = TRUE, ...)

## S4 method for signature 'GbsrGenotypeData'
nscan(object, valid)
```

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
...	Unused.

Details

If valid = TRUE, the number of samples which are labeled TRUE in the ScanAnnotationDataFrame slot will be returned. You need the number of over all samples, set valid = FALSE. getValidSnp() tells you which samples are valid.

Value

An integer value to indicate the number of samples.

See Also

getValidScan()

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

nscan(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

nsnp *Return the number of SNPs.*

Description

This function returns the number of SNPs recorded in the GDS file connected to the given GbsrGenotypeData object.

Usage

```
nsnp(object, valid = TRUE, chr = NULL, ...)

## S4 method for signature 'GbsrGenotypeData'
nsnp(object, valid, chr)
```

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
chr	A index to spefcify chromosome to get information.
...	Unused.

Details

If valid = TRUE, the number of SNPs which are labeled TRUE in the SnpAnnotationDataFrame slot will be returned. You need the number of over all SNPs, set valid = FALSE. getValidSnp() tells you which markers are valid.

Value

An integer value to indicate the number of SNP markers.

See Also

getValidSnp()

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

nsnp(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

openGDS *Open the connection to the GDS file.*

Description

Open the connection to the GDS file.

Usage

```
openGDS(object, ...)

## S4 method for signature 'GbsrGenotypeData'
openGDS(object)
```

Arguments

object	A GbsrGenotypeData object.
...	Unused.

Details

The GbsrGenotypeData object stores the file path of the GDS file even after closing the connection the file. This function open again the connection to the GDS file at the file path stored in the GbsrGenotypeData object. If the GbsrGenotypeData object witch has an open connection to the GDS file, this function will reopen the connection. The data stored in the SnpAnnotationDataFrame and ScanAnnotationDataFrame will not be changed. Thus, you can open a connection with the GDS file with keeping information of filtering and summary statistics.

Value

A GbsrGenotypeData object.

Examples

```
# Use a GDS file of example data.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")

# Instantiation of [GbsrGenotypeData]
gds <- loadGDS(gds_fn)

# Close the connection to the GDS file
closeGDS(gds)

gds <- openGDS(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

---

pairsGBSR *Draw a scatter plot of a pair of specified statistics*

Description

Draw a scatter plot of a pair of specified statistics

Usage

```
pairsGBSR(
  x,
  stats1 = "dp",
  stats2 = "missing",
  target = "snp",
  q = 0.5,
  size = 0.5,
  alpha = 0.8,
  color = c(Marker = "darkblue", Sample = "darkblue"),
  fill = c(Marker = "skyblue", Sample = "skyblue"),
  smooth = FALSE
)
```

Arguments

x	A GbsrGenotypeData object.
stats1	A string to specify statistics to be drawn.
stats2	A string to specify statistics to be drawn.
target	Either or both of "snp" and "scan", e.g. `target = "snp"` to draw a histogram only for SNPs.
q	An integer to specify a quantile calculated via `calcReadStats()`.
size	A numeric value to specify the dot size of a scatter plot.

alpha	A numeric value [0-1] to specify the transparency of dots in a scatter plot.
color	A named vector "Marker" and "Sample" to specify border color of bins in the histograms.
fill	A named vector "Marker" and "Sample" to specify fill color of bins in the histograms.stats = "geno only requires "Ref", "Het" and "Alt", while others uses the value named "Marker".
smooth	A logical value to indicate whether draw a smooth line for data points. See also ggplot2::stat_smooth().

Details

You can draw a scatter plot of per-marker and/or per-sample summary statistics specified at stats1 and stats2. The "stats1" and "stats2" arguments can take the following values:

- "missing""Proportion of missing genotype calls.",
- "het""Proportion of heterozygote calls.",
- "raf""Reference allele frequency.",
- "dp""Total read counts.",
- "ad_ref""Reference allele read counts.",
- "ad_alt""Alternative allele read counts.",
- "rrf""Reference allele read frequency.",
- "mean_ref""Mean of reference allele read counts.",
- "sd_ref""Standard deviation of reference allele read counts.",
- "qtile_ref""Quantile of reference allele read counts.",
- "mean_alt""Mean of alternative allele read counts.",
- "sd_alt""Standard deviation of alternative allele read counts.",
- "qtile_alt""Quantile of alternative allele read counts.",
- "mq""Mapping quality.",
- "fs""Phred-scaled p-value (strand bias)",
- "qd""Variant Quality by Depth",
- "sor""Symmetric Odds Ratio (strand bias)",
- "mqranksum""Alt vs. Ref read mapping qualities",
- "readposranksum""Alt vs. Ref read position bias",
- "baseqranksum""Alt Vs. Ref base qualities",

To draw scatter plots for "missing", "het", "raf", you need to run countGenotype() first to obtain statistics. Similary, "dp", "ad_ref", "ad_alt", "rrf" requires values obtained via countRead(). calcReadStats() should be executed before drawing line plots of "mean_ref", "sd_ref", "qtile_ref", "mean_alt", "sd_alt", and "qtile_alt". "mq", "fs", "qd", "sor", "mqranksum", "readposranksum", and "baseqranksum" only work with target = "snp", if your data contains those values supplied via SNP calling tools like GATK.

Value

A ggplot object.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gdata <- loadGDS(gds_fn)

# Summarize genotype count information to be used in `pairsGBSR()`
gdata <- countGenotype(gdata)

# Draw scatter plots of missing rate vs heterozygosity.
pairsGBSR(gdata, stats1 = "missing", stats2 = "het")

# Close the connection to the GDS file
closeGDS(gdata)
```

---

plotDosage *Draw line plots of allele dosage per marker per sample.*

Description

This function counts a reference allele dosage per marker per sample and draw line plots of them in facets for each chromosome for each sample.

Usage

```
plotDosage(
  x,
  coord = NULL,
  chr = NULL,
  ind = 1,
  node = "raw",
  dot_fill = "green",
  line_color = "magenta"
)
```

Arguments

x	A GbsrGenotypeData object.
coord	A vector with two integer specifying the number of rows and columns to draw faceted line plots for chromosomes.
chr	A vector of indexes to specify chromosomes to be drawn.
ind	An index to specify samples to be drawn.

node	Either one of "raw", "filt", and "cor" to output raw genotype data, filtered genotype data, or corrected genotype data, respectively.
dot_fill	A string to indicate the dot color in the plot.
line_color	A string to indicate the line color in the plot.

Value

A ggplot object.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gdata <- loadGDS(gds_fn)

plotDosage(gdata, ind = 1)

# Close the connection to the GDS file
closeGDS(gdata)
```

---

plotGBSR *Draw line plots of specified statistics*

Description

Draw line plots of specified statistics

Usage

```
plotGBSR(
  x,
  stats = c("dp", "missing", "het"),
  coord = NULL,
  q = 0.5,
  lwd = 0.5,
  binwidth = NULL,
  color = c(Marker = "darkblue", Ref = "darkgreen", Het = "magenta", Alt = "blue")
)
```

Arguments

x	A GbsrGenotypeData object.
stats	A string to specify statistics to be drawn.
coord	A vector with two integer specifying the number of rows and columns to draw faceted line plots for chromosomes.
q	An integer to specify a quantile calculated via calcReadStats().

`lwd`	A numeric value to specify the line width in plots.
`binwidth`	An integer to specify bin width of the histogram. This argument only work with `stats = "marker"` and is passed to the ggplot function.
`color`	A strings vector named "Marker", "Ref", "Het", "Alt" to specify line colors. `stats = "geno` only requires "Ref", "Het" and "Alt", while others uses the value named "Marker".

Details

You can draw line plots of several summary statistics of genotype counts and read counts per sample and per marker. The "stats" argument can take the following values:

- "marker""Marker density.",
- "geno""Proportion of missing genotype calls.",
- "missing""Proportion of missing genotype calls.",
- "het""Proportion of heterozygote calls.",
- "raf""Reference allele frequency.",
- "dp""Total read counts.",
- "ad_ref""Reference allele read counts.",
- "ad_alt""Alternative allele read counts.",
- "rrf""Reference allele read frequency.",
- "mean_ref""Mean of reference allele read counts.",
- "sd_ref""Standard deviation of reference allele read counts.",
- "qtile_ref""Quantile of reference allele read counts.",
- "mean_alt""Mean of alternative allele read counts.",
- "sd_alt""Standard deviation of alternative allele read counts.",
- "qtile_alt""Quantile of alternative allele read counts.",
- "mq""Mapping quality.",
- "fs""Phred-scaled p-value (strand bias)",
- "qd""Variant Quality by Depth",
- "sor""Symmetric Odds Ratio (strand bias)",
- "mqranksum""Alt vs. Ref read mapping qualities",
- "readposranksum""Alt vs. Ref read position bias",
- "baseqranksum""Alt Vs. Ref base qualities",

To draw line plots for "missing", "het", "raf", you need to run `countGenotype()` first to obtain statistics. Similary, "dp", "ad_ref", "ad_alt", "rrf" requires values obtained via `countRead()`. `calcReadStats()` should be executed before drawing line plots of "mean_ref", "sd_ref", "qtile_ref", "mean_alt", "sd_alt", and "qtile_alt". "mq", "fs", "qd", "sor", "mqranksum", "readposranksum", #' and "baseqranksum" only work with `target = "snp"`, if your data contains those values supplied via SNP calling tools like GATK.

Value

A ggplot object.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gdata <- loadGDS(gds_fn)

# Summarize genotype count information to be used in `plotGBSR()`
gdata <- countGenotype(gdata)

# Draw line plots of missing rate, heterozygosity, proportion of genotype
# calls per SNP.
plotGBSR(gdata, stats = "missing")

# Calculate means, standard deviations, quantile values of read counts
# to be used in `plotGBSR()`
gdata <- calcReadStats(gdata, q = 0.9)

# Draw line plots of 90 percentile values of reference read counts and
# alternative read counts per SNP and per sample.
plotGBSR(gdata, stats = "qtile_ref", q = 0.9)

# Close the connection to the GDS file
closeGDS(gdata)
```

---

plotReadRatio *Draw line plots of proportion of reference allele read counts per marker per sample.*

Description

This function calculate a proportion of reference allele read counts per marker per sample and draw line plots of them in facets for each chromosome for each sample.

Usage

```
plotReadRatio(
  x,
  coord = NULL,
  chr = NULL,
  ind = 1,
  node = "raw",
  dot_fill = "blue"
)
```

Arguments

x	A GbsrGenotypeData object.
coord	A vector with two integer specifying the number of rows and columns to draw faceted line plots for chromosomes.
chr	A vector of indexes to specify chromosomes to be drawn.
ind	An index to specify samples to be drawn.
node	Either one of "raw", "filt", and "cor" to output raw genotype data, filtered genotype data, or corrected genotype data, respectively.
dot_fill	A string to indicate the dot color in a plot.

Value

A ggplot object.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gdata <- loadGDS(gds_fn)

plotReadRatio(gdata, ind = 1)

# Close the connection to the GDS file
closeGDS(gdata)
```

---

resetFilters *Reset all filters made by* setScanFilter(), setSnpFilter(), *and* setCallFilter().

Description

Return all data intact.

Usage

```
resetFilters(object, ...)

## S4 method for signature 'GbsrGenotypeData'
resetFilters(object)
```

Arguments

object	A GbsrGenotypeData object.
...	Unused.

Value

A GbsrGenotypeData object after removing all filters.

A GbsrGenotypeData object after removing all filters on markers.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# `setCallFilter()` do not require summarized information of
# genotype counts and read counts.
gds <- setCallFilter(gds, dp_count = c(5, Inf))

# `setScanFilter()` and `setSnpFilter()` needs information of
# the genotype count summary and the read count summary.
gds <- countGenotype(gds)
gds <- countRead(gds)

gds <- setScanFilter(gds,
                     id = getScanID(gds)[1:10],
                     missing = 0.2,
                     dp = c(5, Inf))

gds <- setSnpFilter(gds,
                    id = getSnpID(gds)[1:100],
                    missing = 0.2,
                    dp = c(5, Inf))

gds <- setInfoFilter(gds, mq = 40, qd = 20)

# Reset all filters applied above.
gds <- resetFilters(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

resetScanFilters *Reset the filter made by* setScanFilter()

Description

Remove "invalid" labels put on samples and make all samples valid.

Usage

```
resetScanFilters(object, ...)

## S4 method for signature 'GbsrGenotypeData'
resetScanFilters(object)
```

Arguments

object	A GbsrGenotypeData object.
...	Unused.

Value

A GbsrGenotypeData object after removing all filters on scan(samples).

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Summarize the information needed for filtering.
gds <- countGenotype(gds)
gds <- countRead(gds)

gds <- setScanFilter(gds,
                     id = getScanID(gds)[1:10],
                     missing = 0.2,
                     dp = c(5, Inf))

# Reset all filters applied above.
gds <- resetScanFilters(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

---

resetSnpFilters *Reset the filter made by* setSnpFilter()

Description

Remove "invalid" labels put on markers and make all markers valid.

Usage

```
resetSnpFilters(object, ...)

## S4 method for signature 'GbsrGenotypeData'
resetSnpFilters(object)
```

Arguments

object	A GbsrGenotypeData object.
...	Unused.

Value

A GbsrGenotypeData object after removing all filters on markers.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Check the number of markers.
nsnp(gds)

# Summarize the information needed for filtering.
gds <- countGenotype(gds)
gds <- countRead(gds)

# filter out some markers meeting the criteria.
gds <- setSnpFilter(gds,
                    id = getSnpID(gds)[1:100],
                    missing = 0.2,
                    dp = c(5, Inf))

# Check the number of the retained markers.
nsnp(gds)

# Reset all filters applied above.
gds <- resetSnpFilters(gds)

# Check the number of the markers again.
nsnp(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

saveScanAnnot *Write out the information stored in the ScanAnnotationDataFrame slot*

Description

All the data stored in a ScanAnnotationDataFrame slot of the GbsrGenotypeData object can be saved in the GDS file linked to the given GbsrGenotypeData object. You can load the saved data using `loadSnpAnnot()`.

Usage

```
saveScanAnnot(object, ...)

## S4 method for signature 'GbsrGenotypeData'
saveScanAnnot(object)
```

Arguments

`object`	A GbsrGenotypeData object
`...`	Unused.

Value

NA GbsrGenotypeData object.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Save data stored in a [ScanAnnotationDataFrame] object
# linked in the slot of the `GbsrGenotypeData.`
gds <- saveSnpAnnot(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

saveSnpAnnot *Write out the information stored in the SnpAnnotationDataFrame slot*

Description

All the data stored in the SnpAnnotatoinDataFrame slot of the GbsrGenotypeData object can be saved in the GDS file linked to the given GbsrGenotypeData object. You can load the saved data using `loadSnpAnnot()`.

Usage

```
saveSnpAnnot(object, ...)

## S4 method for signature 'GbsrGenotypeData'
saveSnpAnnot(object)
```

Arguments

object	A GbsrGenotypeData object
...	Unused.

Value

A GbsrGenotypeData object.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Save data stored in the [SnpAnnotationDataFrame] object
# linked in the slot of the `GbsrGenotypeData.`
gds <- saveSnpAnnot(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

`setCallFilter` *Filter out each genotype call meeting criteria*

Description

Perform filtering of each genotype call, neither markers nor samples. Each genotype call is supported by its read counts for the reference allele and the alternative allele of a marker of a sample. `setCallFilter()` set missing to the genotype calls which are not reliable enough and set zero to reference and alternative read counts of the genotype calls.

Usage

```
setCallFilter(
  object,
  dp_count = c(0, Inf),
  ref_count = c(0, Inf),
  alt_count = c(0, Inf),
  norm_dp_count = c(0, Inf),
  norm_ref_count = c(0, Inf),
  norm_alt_count = c(0, Inf),
  scan_dp_qtile = c(0, 1),
  scan_ref_qtile = c(0, 1),
  scan_alt_qtile = c(0, 1),
  snp_dp_qtile = c(0, 1),
  snp_ref_qtile = c(0, 1),
  snp_alt_qtile = c(0, 1),
  ...
)

## S4 method for signature 'GbsrGenotypeData'
setCallFilter(
  object,
  dp_count,
  ref_count,
  alt_count,
  norm_dp_count,
  norm_ref_count,
  norm_alt_count,
  scan_dp_qtile,
  scan_ref_qtile,
  scan_alt_qtile,
  snp_dp_qtile,
  snp_ref_qtile,
  snp_alt_qtile
)
```

Arguments

object	A GbsrGenotypeData object.
dp_count	A numeric vector with length two specifying lower and upper limit of total read counts (reference reads + alternative reads).
ref_count	A numeric vector with length two specifying lower and upper limit of reference read counts.
alt_count	A numeric vector with length two specifying lower and upper limit of alternative read counts.
norm_dp_count	A numeric vector with length two specifying lower and upper limit of normalized total read counts (normalized reference reads + normalized alternaitve reads).
norm_ref_count	A numeric vector with length two specifying lower and upper limit of normalized reference read counts.
norm_alt_count	A numeric vector with length two specifying lower and upper limit of normalized alternative read counts
scan_dp_qtile	A numeric vector with length two specifying lower and upper limit of quantile of total read counts in each scan (sample).
scan_ref_qtile	A numeric vector with length two specifying lower and upper limit of quantile of reference read counts in each scan (sample).
scan_alt_qtile	A numeric vector with length two specifying lower and upper limit of quantile of alternative read counts in each scan (sample).
snp_dp_qtile	A numeric vector with length two specifying lower and upper limit of quantile of total read counts in each SNP marker
snp_ref_qtile	A numeric vector with length two specifying lower and upper limit of quantile of reference read counts in each SNP marker.
snp_alt_qtile	A numeric vector with length two specifying lower and upper limit of quantile of alternative read counts in each SNP marker.
...	Unused.

Details

norm_dp_count, norm_ref_count, and norm_alt_count use normalized read counts which are obtained by dividing each read count by the total read count of each sample. scan_dp_qtile, scan_ref_qtile, and scan_alt_qtile work similarly but use quantile values of read counts of each sample to decide the lower and upper limit of read counts. This function generate two new nodes in the GDS file linked with the given GbsrGenotypeData object. The new nodes "filt.data" in the AD node and "filt.genotype" contains read count data and genotype data after filtering, respectively. To reset the filter applied by setCallFilter(), run `setRawGenotype()`.

Value

A GbsrGenotypeData object with filters on genotype calls.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Filter out genotype calls supported by less than 5 reads.
gds <- setCallFilter(gds, dp_count = c(5, Inf))

# Filter out genotype calls supported by reads less than
# the 20 percentile of read counts per marker in each sample.
gds <- setCallFilter(gds, scan_dp_qtile = c(0.2, 1))

# Reset the filter
gds <- setRawGenotype(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

---

setFiltGenotype *Set the filtered data to be used in GBScleanR's functions*

Description

Set the "filt.genotype" node and the "filt.data" node as primary nodes for genotype data and read count data. The data stored in the primary nodes are used in the functions of GBScleanR.

Usage

```
setFiltGenotype(object, ...)

## S4 method for signature 'GbsrGenotypeData'
setFiltGenotype(object)
```

Arguments

`object`	A GbsrGenotypeData object.
`...`	Unused.

Details

A GbsrGenotypeData object storing information of the primary node of genotype data and read count data. All of the functions implemented in GBScleanR check the primary nodes and use data stored in those nodes. `setCallFilter()` create new nodes storing filtered genotype calls and read

counts in a GDS file and change the primary nodes to "filt.genotype" and "filt.data" for genotype and read count data, respectively. setRawGenotype() set back the nodes to the original, those are "genotype" and "data" for genotype and read count data, respectively. You can set the filtered data again by setFiltGenotype().

Value

A GbsrGenotypeData object.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Filter out set zero to read counts and
# missing to genotype calls of which meet the criteria.
gds <- setCallFilter(gds, dp_count = c(5, Inf))

# Now any functions of [GBScleanR] reference the genotype data
# stored in the "filt.genotype" node of the GDS file.

# Close and reopen the connection to the GDS file.
closeGDS(gds)
gds <- loadGDS(gds)

# If the connection was closed once and re-loaded,
# raw genotype data is referenced by the functions.

# To set the "filt.genotype" node as genotype data again,
# run the setFiltGenotype().
gds <- setFiltGenotype(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

---

setInfoFilter *Filter out markers based on marker quality metrics*

Description

A VCF file usually has marker quality metrics in the INFO filed and those are stored in a GDS file created via GBScleanR. This function filter out markers based on those marker quality metrics.

Usage

```
setInfoFilter(
  object,
  mq = 0,
  fs = Inf,
  qd = 0,
  sor = Inf,
  mqranksum = c(-Inf, Inf),
  readposranksum = c(-Inf, Inf),
  baseqranksum = c(-Inf, Inf),
  ...
)

## S4 method for signature 'GbsrGenotypeData'
setInfoFilter(object, mq, fs, qd, sor, mqranksum, readposranksum, baseqranksum)
```

Arguments

object	A GbsrGenotypeData object.
mq	A numeric value to specify minimum mapping quality (shown as MQ in the VCF format).
fs	A numeric value to specify maximum Phred-scaled p-value (strand bias) (shown as FS in the VCF format).
qd	A numeric value to specify minimum Variant Quality by Depth (shown as QD in the VCF format).
sor	A numeric value to specify maximum Symmetric Odds Ratio (strand bias) (shown as SOR in the VCF format).
mqranksum	A numeric values to specify the lower and upper limit of Alt vs. Ref read mapping qualities (shown as MQRankSum in the VCF format).
readposranksum	A numeric values to specify the lower and upper limit of Alt vs. Ref read position bias (shown as ReadPosRankSum in the VCF format).
baseqranksum	A numeric values to specify the lower and upper limit of Alt Vs. Ref base qualities (shown as BaseQRankSum in the VCF format).
...	Unused.

Details

Detailed explanation of each metric can be found in GATK's web site.

Value

A GbsrGenotypeData object with filters on markers.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

gds <- setInfoFilter(gds, mq = 40, qd = 20)

# Close the connection to the GDS file.
closeGDS(gds)
```

`setParents` *Set labels to samples which should be recognized as parents of the population to be subjected to error correction.*

Description

Specify two or more samples in the dataset as parents of the population. Markers will be filtered out up on your specification.

Usage

```
setParents(object, parents, flip = FALSE, mono = FALSE, bi = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
setParents(object, parents, flip, mono, bi)
```

Arguments

`object`	A GbsrGenotypeData object.
`parents`	A vector of strings with at least length two. The specified strings should match with the samples ID available via `getScanID()`.
`flip`	A logical value to indicate whether markers should be checked for "flip". See details.
`mono`	A logical value whether to filter out markers which are not monomorphic in parents.
`bi`	A logical value whether to filter out marekrs which are not biallelic between parents.
`...`	Unused.

Details

The `clean` function of GBScleanR uses read count information of samples and their parents separately to estimate most probable genotype calls of them. Therefore, you must specify proper samples as parents via this function. If you would like to remove SNP markers which are not biallelic and/or not monomorphic in each parent, set `mono = TRUE` and `bi = TRUE`. `flip = TRUE` flips alleles of markers where the alleles expected as reference allele are called as alternative allele. The

alleles found in the parent specified as the first element to the `parents` argument are supposed as reference alleles of the markers. If the "expected" reference alleles are not actually called as reference alleles but alternative alleles in the given data. setParents() will automatically labels those markers "flipped". The SnpAnnotatoinDataFrame slot sores this information and accessible via `getFlipped()` which gives you a logical vector indicating which markers are labeled as flipped `TRUE` or not flipped `FALSE`. `hasFlipped()` just tells you whether the SnpAnnotatoinDataFrame slot has the information of flipped markers or not.

Value

A GbsrGenotypeData object with parents information.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Find the IDs of parental samples.
parents <- grep("Founder", getScanID(gds), value = TRUE)

# Set the parents and flip allele information
# if the reference sample (Founder1 in our case) has homozygous
# alternative genotype at some markers of which alleles will
# be swapped to make the reference sample have homozygous
# reference genotype.
gds <- setParents(gds, parents = parents, flip = TRUE)

# Initialize a scheme object stored in the slot of the GbsrGenotypeData.
# We chose `crosstype = "pair"` because two inbred founders were mated
# in our breeding scheme.
# We also need to specify the mating matrix which has two rows and
# one column with integers 1 and 2 indicating a sample (founder)
# with the memberID 1 and a sample (founder) with the memberID 2
# were mated.
gds <- initScheme(gds, crosstype = "pair", mating = cbind(c(1:2)))

# Add information of the next cross conducted in our scheme.
# We chose 'crosstype = "selfing"', which do not require a
# mating matrix.
gds <- addScheme(gds, crosstype = "selfing")

# Execute error correction by estimating genotype and haplotype of
# founders and offspring.
gds <- estGeno(gds)

# Close the connection to the GDS file.
closeGDS(gds)
```

setRawGenotype *Set the origina; data to be used in GBScleanR's functions*

Description

Set the "genotype" node and the "data" node as primary nodes for genotype data and read count data. The data stored in the primary nodes are used in the functions of GBScleanR.

Usage

```
setRawGenotype(object, ...)

## S4 method for signature 'GbsrGenotypeData'
setRawGenotype(object)
```

Arguments

object	A GbsrGenotypeData object.
...	Unused.

Details

A GbsrGenotypeData object storing information of the primary node of genotype data and read count data. All of the functions implemented in GBScleanR check the primary nodes and use data stored in those nodes. `setCallFilter()` create new nodes storing filtered genotype calls and read counts in a GDS file and change the primary nodes to "filt.genotype" and "filt.data" for genotype and read count data, respectively. `setRawGenotype()` set back the nodes to the original, those are "genotype" and "data" for genotype and read count data, respectively. You can set the filtered data again by `setFiltGenotype()`.

Value

A GbsrGenotypeData object.

Examples

```
# Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Filter out set zero to read counts and
# missing to genotype calls of which meet the criteria.
gds <- setCallFilter(gds, dp_count = c(5, Inf))
```

```
# Now any functions of [GBScleanR] reference the genotype data
# stored in the "filt.genotype" node of the GDS file.

# If you need to set the "genotype" node, where store the raw genotype data
# as genotype to be referenced by the functions of GBScleanR,
# run the following.
gds <- setRawGenotype(gds)

# Reopening the connection to the GDS file also set the raw genotype again.
gds <- loadGDS(gds)

# Close the connection to the GDS file
closeGDS(gds)
```

`setScanFilter` *Filter out scans (samples)*

Description

Search samples which do not meet the criteria and label them as "invalid".

Usage

```
setScanFilter(
  object,
  id = "",
  missing = 1,
  het = c(0, 1),
  mac = 0,
  maf = 0,
  ad_ref = c(0, Inf),
  ad_alt = c(0, Inf),
  dp = c(0, Inf),
  mean_ref = c(0, Inf),
  mean_alt = c(0, Inf),
  sd_ref = Inf,
  sd_alt = Inf,
  ...
)

## S4 method for signature 'GbsrGenotypeData'
setScanFilter(
  object,
  id,
  missing,
  het,
  mac,
  maf,
```

```
  ad_ref,
  ad_alt,
  dp,
  mean_ref,
  mean_alt,
  sd_ref,
  sd_alt
)
```

Arguments

object	A GbsrGenotypeData object.
id	A vector of strings matching with scan ID which can be retrieve by getScanID(). The samples with the specified IDs will be filtered out.
missing	A numeric value [0-1] to specify the maximum missing genotype call rate per sample.
het	A vector of two numeric values [0-1] to specify the minimum and maximum heterozygous genotype call rate per sample.
mac	A integer value to specify the minimum minor allele count per sample.
maf	A numeric value to specify the minimum minor allele frequency per sample.
ad_ref	A numeric vector with length two specifying lower and upper limit of reference read counts per sample.
ad_alt	A numeric vector with length two specifying lower and upper limit of alternative read counts per sample.
dp	A numeric vector with length two specifying lower and upper limit of total read counts per sample.
mean_ref	A numeric vector with length two specifying lower and upper limit of mean of reference read counts per sample.
mean_alt	A numeric vector with length two specifying lower and upper limit of mean of alternative read counts per sample.
sd_ref	A numeric value specifying the upper limit of standard deviation of reference read counts per sample.
sd_alt	A numeric value specifying the upper limit of standard deviation of alternative read counts per sample.
...	Unused.

Details

For mean_ref, mean_alt, sd_ref, and sd_alt, this function calculate mean and standard deviation of reads obtained at SNP markers of each sample. If a mean read counts of a sample was smaller than the specified lower limit or larger than the upper limit, this function labels the sample as "invalid". In the case of sd_ref and sd_alt, standard deviations of read counts of each sample are checked and the samples having a larger standard deviation will be labeled as "invalid". To check valid and invalid samples, run getValidScan().

Value

A GbsrGenotypeData object with filters on scan(samples).

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the information needed for filtering.
gds <- countGenotype(gds)
gds <- countRead(gds)

gds <- setScanFilter(gds,
                     id = getScanID(gds)[1:10],
                     missing = 0.2,
                     dp = c(5, Inf))

# Close the connection to the GDS file.
closeGDS(gds)
```

`setSnpFilter` *Filter out markers*

Description

Search markers which do not meet the criteria and label them as "invalid".

Usage

```
setSnpFilter(
  object,
  id = NA_integer_,
  missing = 1,
  het = c(0, 1),
  mac = 0,
  maf = 0,
  ad_ref = c(0, Inf),
  ad_alt = c(0, Inf),
  dp = c(0, Inf),
  mean_ref = c(0, Inf),
  mean_alt = c(0, Inf),
  sd_ref = Inf,
  sd_alt = Inf,
  ...
)
```

```
## S4 method for signature 'GbsrGenotypeData'
setSnpFilter(
  object,
  id,
  missing,
  het,
  mac,
  maf,
  ad_ref,
  ad_alt,
  dp,
  mean_ref,
  mean_alt,
  sd_ref,
  sd_alt
)
```

Arguments

object	A GbsrGenotypeData object.
id	A vector of integers matching with snp ID which can be retrieve by getSnpID(). The markers with the specified IDs will be filtered out.
missing	A numeric value [0-1] to specify the maximum missing genotype call rate per marker
het	A numeric vector with length two [0-1] to specify the minimum and maximum heterozygous genotype call rate per marker
mac	A integer value to specify the minimum minor allele count per marker
maf	A numeric value to specify the minimum minor allele frequency per marker.
ad_ref	A numeric vector with length two specifying lower and upper limit of reference read counts per marker.
ad_alt	A numeric vector with length two specifying lower and upper limit of alternative read counts per marker.
dp	A numeric vector with length two specifying lower and upper limit of total read counts per marker.
mean_ref	A numeric vector with length two specifying lower and upper limit of mean of reference read counts per marker.
mean_alt	A numeric vector with length two specifying lower and upper limit of mean of alternative read counts per marker.
sd_ref	A numeric value specifying the upper limit of standard deviation of reference read counts per marker.
sd_alt	A numeric value specifying the upper limit of standard deviation of alternative read counts per marker.
...	Unused.

Details

For mean_ref, mean_alt, sd_ref, and sd_alt, this function calculate mean and standard deviation of reads obtained for samples at each SNP marker. If a mean read counts of a marker was smaller than the specified lower limit or larger than the upper limit, this function labels the marker as "invalid". In the case of sd_ref and sd_alt, standard deviations of read counts of each marker are checked and the markers having a larger standard deviation will be labeled as "invalid". To check valid and invalid markers, run getValidSnp().

Value

A GbsrGenotypeData object with filters on markers.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize the information needed for filtering.
gds <- countGenotype(gds)
gds <- countRead(gds)

gds <- setSnpFilter(gds,
                    id = getSnpID(gds)[1:100],
                    missing = 0.2,
                    dp = c(5, Inf))

# Close the connection to the GDS file.
closeGDS(gds)
```

---

setValidScan *Manually set valid scans (samples).*

Description

If you need manually set valid and invalid samples, you can do it via this function, e.g in the case you conducted a filtering on samples manually by your self.

Usage

```
setValidScan(object, new, update, ...)

## S4 method for signature 'GbsrGenotypeData'
setValidScan(object, new, update)
```

Arguments

object	A GbsrGenotypeData object.
new	A logical vector of the same length with the over all number of the samples.
update	A logical vector of the same length with the currently valid samples.
...	Unused.

Details

To over write the current validity information, give a logical vector to new. On the other hand, a logical vector specified to update will be used to update validity information of the currently valid samples. If you gave a vector for both argument, only the vector passed to new will be used to over write the validity information.

Value

A GbsrGenotypeData object with updated scan(sample) validity information.

See Also

setScanFilter() to filter out samples based on some summary statistics

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Randomly remove samples.
gds <- setValidScan(gds,
                    update = sample(c(TRUE, FALSE),
                                    nscan(gds),
                                    replace = TRUE))

# Close the connection to the GDS file.
closeGDS(gds)
```

---

setValidSnp *Manually set valid SNP markers.*

Description

If you need manually set valid and invalid SNP markers, you can do it via this function, e.g in the case you conducted a filtering on SNP markers manually by your self.

Usage

```
setValidSnp(object, new, update, ...)

## S4 method for signature 'GbsrGenotypeData'
setValidSnp(object, new, update)
```

Arguments

object	A GbsrGenotypeData object.
new	A logical vector of the same length with the over all number of the SNP markers.
update	A logical vector of the same length with the currently valid SNP markers.
...	Unused.

Details

To over write the current validity information, give a logical vector to new. On the other hand, a logical vector specified to update will be used to update validity information of the currently valid SNP markers. If you gave a vector for both argument, only the vector passed to new will be used to over write the validity information.

Value

A GbsrGenotypeData object with updated SNP validity information.

See Also

setSnpFilter() to filter out SNP markers based on some summary statistics.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Randomly remove SNP markers.
gds <- setValidSnp(gds,
                   update = sample(c(TRUE, FALSE),
                                   nsnp(gds),
                                   replace = TRUE))

# Close the connection to the GDS file.
closeGDS(gds)
```

showScheme *Show the information stored in a GbsrScheme object*

Description

Print the information of each generation in a GbsrScheme object in the scheme slot of a GbsrGenotypeData object. A GbsrScheme object stores information of a population size, mating combinations and a type of cross applied to each generation of the breeding process to generate the population which you are going to subject to the `estGeno()` function.

Usage

```
showScheme(object, ...)

## S4 method for signature 'GbsrGenotypeData'
showScheme(object)

## S4 method for signature 'GbsrScheme'
showScheme(object, parents_name)
```

Arguments

`object`	A GbsrGenotypeData object.
`...`	Unused.
`parents_name`	A vector of strings to indicate names of parental samples.

Value

NULL. Print the scheme information on the R console.

See Also

`initScheme()` and `addScheme()`

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Biparental F2 population.
gds <- setParents(gds, parents = c("Founder1", "Founder2"))

# setParents gave member ID 1 and 2 to Founder1 and Founder2, respectively.
gds <- initScheme(gds, crosstype = "pair", mating = cbind(c(1:2)))

# Now the progenies of the cross above have member ID 3.
# If `crosstype = "selfing"` or `"sibling"`, you can omit a `mating` matrix.
```

```
gds <- addScheme(gds, crosstype = "self")

# Now you can execute `estGeno()` which requires a [GbsrScheme] object.

# Close the connection to the GDS file
closeGDS(gds)
```

---

subsetGDS *Create a GDS file with subset data of the current GDS file*

Description

Create a new GDS file storing the subset data from the current GDS file linked to the given GbsrGenotypeData object with keeping (or removing) information based on valid markers and samples information.

Usage

```
subsetGDS(
  object,
  out_fn = "./susbet.gds",
  snp_incl,
  scan_incl,
  incl_parents = TRUE,
  verbose = TRUE,
  ...
)

## S4 method for signature 'GbsrGenotypeData'
subsetGDS(object, out_fn, snp_incl, scan_incl, incl_parents, verbose)
```

Arguments

object	A GbsrGenotypeData object.
out_fn	A string to specify the path to an output GDS file.
snp_incl	A logical vector having the same length with the total number of markers. The values obtained via `getValidSnp()` are used.
scan_incl	A logical vector having the same length with the total number of scans (samples). The values obtained via `getValidScan()` are used.
incl_parents	A logical value to specify whether parental samples should be included in a subset data or not.
verbose	if TRUE, show information.
...	Unused.

Details

A resultant subset data in a new GDS file includes subsets of each category of data, e.g. genotype, SNP ID, scan ID, read counts, and quality metrics of SNP markers. The connection to the GDS file of the input GbsrGenotypeData object will be automatically closed for internal file handling in this function. Please use `openGDS()` to open the connection again. If you use `loadGDS()`, summary statistics and filtering information will be discarded.

Value

A GbsrGenotypeData object linking to the new GDS file storing subset data.

Examples

```
#' # Create a GDS file from a sample VCF file.
vcf_fn <- system.file("extdata", "sample.vcf", package = "GBScleanR")
gds_fn <- tempfile("sample", fileext = ".gds")
gbsrVCF2GDS(vcf_fn = vcf_fn, out_fn = gds_fn, force = TRUE)

# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds <- loadGDS(gds_fn)

# Summarize genotype count information to be used in `setSnpFilter()`
gds <- countGenotype(gds)

# Filter out markers meeting the criteia.
gds <- setSnpFilter(gds, missing = 0.2, het = c(0.1, 0.9), maf = 0.05)

# Create a new GDS file with the subset data obtained by applying
# the filter maed via `setSnpFilter()`.
subsetgds_fn <- tempfile("sample_subset", fileext = ".gds")
subset_gds <- subsetGDS(gds, out_fn = subsetgds_fn)

# Close the connection to the GDS files.
closeGDS(subset_gds)
```

---

thinMarker *Remove markers potentially having redundant information.*

Description

Markers within the length of the sequenced reads (usually ~ 150 bp, up to your sequencer) potentially have redundant information and those will cause unexpected errors in error correction which assumes independency of markers each other. This function only retains the first marker or the least missing rate marker from the markers locating within the specified stretch.

Usage

```
thinMarker(object, range = 150, ...)

## S4 method for signature 'GbsrGenotypeData'
thinMarker(object, range)
```

Arguments

object	A GbsrGenotypeData object.
range	A integer value to indicate the stretch to search markers.
...	Unused.

Details

This function search valid markers from the first marker of each chromosome and compare its physical position with a neighbor marker. If the distance between those markers are equal or less then range, one of them which has a larger missing rate will be removed (labeled as invalid marker). When the first marker was retained and the second marker was removed as invalid marker, next the distance between the first marker and the third marker will be checked and this cycle is repeated until reaching the end of each chromosome. Run `getValidSnp()` to check the valid SNP markers.

Value

A GbsrGenotypeData object with filters on markers.

Examples

```
# Load data in the GDS file and instantiate a [GbsrGenotypeData] object.
gds_fn <- system.file("extdata", "sample.gds", package = "GBScleanR")
gds <- loadGDS(gds_fn)

# Summarize genotype count information to be used in thinMarker().
gds <- countGenotype(gds)
gds <- thinMarker(gds, range = 150)

closeGDS(gds) # Close the connection to the GDS file
```

Index