Package 'GBScleanR'

November 13, 2026

Type Package

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

Version 2.7.5

Date 2026-10-09

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

Depends SeqArray

Imports stats, utils, methods, ggplot2, tidyr, expm, Rcpp, RcppParallel, gdsfmt

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

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

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

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

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

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

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

git_branch devel

git_last_commit d9511e9

git_last_commit_date 2026-10-09

Repository Bioconductor 3.24

Date/Publication 2026-11-12

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

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

Contents

addScheme . . . 3
assignScheme . . . 5
boxplotGBSR . . . 6
closeGDS . . . 8
countGenotype . . . 9
countRead . . . 10
estGeno . . . 11
GBScleanR . . . 13
gbsrGDS2CSV . . . 14
gbsrGDS2VCF . . . 15
GbsrGenotypeData-class . . . 17
GbsrScheme-class . . . 17
gbsrVCF2GDS . . . 18
getAllele . . . 19
getChromosome . . . 20
getCountAlleleAlt . . . 21
getCountAlleleMissing . . . 22
getCountAlleleRef . . . 23
getCountGenoAlt . . . 24
getCountGenoHet . . . 25
getCountGenoMissing . . . 26
getCountGenoRef . . . 27
getCountRead . . . 28
getCountReadAlt . . . 29
getCountReadRef . . . 30
getFixedParameter . . . 31
getGenotype . . . 32
getHaplotype . . . 33
getInfo . . . 35
getMAC . . . 36
getMAF . . . 37
getMarID . . . 38
getMeanReadAlt . . . 39
getMeanReadRef . . . 40
getMedianReadAlt . . . 41
getMedianReadRef . . . 42
getParents . . . 43

getPloidy . . . 44
getPosition . . . 44
getRead . . . 45
getReplicates . . . 46
getSamID . . . 48
getSDReadAlt . . . 49
getSDReadRef . . . 50
histGBSR . . . 51
initScheme . . . 52
isOpenGDS . . . 54
loadGDS . . . 55
makeScheme . . . 56
nmar . . . 57
nsam . . . 58
pairsGBSR . . . 59
plotDosage . . . 61
plotGBSR . . . 62
plotReadRatio . . . 64
reopenGDS . . . 65
resetCallFilter . . . 66
resetFilter . . . 67
resetMarFilter . . . 68
resetSamFilter . . . 69
setCallFilter . . . 70
setFixedParameter . . . 72
setInfoFilter . . . 73
setMarFilter . . . 74
setParents . . . 76
setPloidy . . . 78
setReplicates . . . 79
setSamFilter . . . 80
showScheme . . . 82
thinMarker . . . 83
validMar . . . 84
validSam . . . 85

Index **87**

`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, ...)

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

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

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.
...	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, 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)
```

`assignScheme` *Assign member IDs to samples*

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 assign member IDs to indicate which samples were derived from which pedigree that recorded in the GbsrScheme object.

Usage

```
assignScheme(object, id, ...)

## S4 method for signature 'GbsrGenotypeData'
assignScheme(object, id)

## S4 method for signature 'GbsrScheme'
assignScheme(object, id)
```

Arguments

`object`	A GbsrGenotypeData object.
`id`	A numeric vector indicating member IDs to assign to samples.
`...`	Unused.

Details

Member IDs can be shown by `showScheme()`. Only the member IDs assigned to progenies (not parents) are available to assign to samples. If the last generation recorded in the GbsrScheme object has only one member ID that should be assigned to all samples in your population, you can omit assigning IDs by `assignScheme()`. In that case, `estGeno()` automatically assign the only one member ID to all samples.

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, mating = cbind(c(1:2)))

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

# The progeny of the selfing above has member ID 4.
# To execute genotype estimation for your samples, you need to assign a member
# ID to each of the samples.

# Check IDs of samples to be assigned member IDs if necessary.
getSamID(gds)

# The assignScheme() assign member IDs `id` to the samples in order.
# Please confirm the order of the member IDs in `id` and the order of the
# sample IDs shown by getSamID(gds).
gds <- assignScheme(gds, rep(4, nsam(gds)))

# If your population has samples all of which belong to only one pedigree,
# you can omit assignScheme() and let estGeno() automatically assign the
# last member ID to all samples.

################################################################################
# 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("marker", "sample"),
  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 "marker" and "sample", e.g. target = "marker" to draw a histogram only for SNPs.
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.

median_ref Quantile of reference allele read counts.

mean_alt Mean of alternative allele read counts.

sd_alt Standard deviation of alternative allele read counts.

median_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(). "mq", "fs", "qd", "sor", "mqranksum", "readposranksum", and "baseqranksum" only work with target = "marker", 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")
gds <- loadGDS(gds_fn)

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

boxplotGBSR(gds, stats = "missing")

# 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, save_filter = FALSE, verbose = TRUE, ...)

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

Arguments

object	A GbsrGenotypeData object.
save_filter	A logical whether to save the filtering information made via setSamFilter() and setMarFilter() in the GDS file. The saved filter information can be reused if set load_filter = TRUE for loadGDS().
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 sample 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 "marker" and "sample".
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
# [marker] and [sample] slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

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

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

```
# 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 sample slot and obtained via getter functions, e.g. `getCountReadRef()` and `getCountReadAlt()`. 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 sample slot and obtained via getter functions, e.g. `getMeanReadRef()` and `getMedianReadAlt()`.

Usage

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

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

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "marker" and "sample".
`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
# [marker] and [sample] slots of the [GbsrGenotypeData] object.
gds <- countRead(gds)

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

# Get the proportion of reference allele rads per marker.
reference_read_freq <- getCountReadRef(gds, target = "marker", 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,
  node = "raw",
  recomb_rate = 0.04,
  error_rate = 0.0025,
  call_threshold = 0.9,
  het_parent = FALSE,
  optim = TRUE,
  iter = 4,
  n_threads = 1,
  dummy_reads = 5,
  ...
)

## S4 method for signature 'GbsrGenotypeData'
estGeno(
  object,
  node,
  recomb_rate,
  error_rate,
```

```
  call_threshold,
  het_parent,
  optim,
  iter,
  n_threads,
  dummy_reads
)
```

Arguments

object	A GbsrGenotypeData object.
node	Either "raw" or "filt" to indicate whether raw or filtered read counts are used for genotype estimation. See setCallFilter() for the details of filtered read counts.
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 1 and if n_threads = NULL, automatically set half the number of available threads on the computer.
dummy_reads	An integer to specify the number of dummy reads to assign to dummy parental samples for genotype estimation. See details.
...	Unused.

Details

If you have not set parental samples by setParents() and initialized the scheme object using initScheme(), you have the scheme object without explicit parental information that is assumed to be a bi-parental population. In this case, estGeno() will run in the parentless mode. In the parentless mode, the algorithm assumes that the given population is a bi-parental population. The number of reference allele reads and the number of alternative allele reads of the dummy parents are set based on dummy_reads, respectively. Dummy parent 1 has dummy_reads of the reference allele reads and 0 alternative allele reads at all markers, while dummy parent 2 has 0 and dummy_reads of reference and alternative allele reads at all markers. If the parents of your population were outbred lines or you cannot assume one of the parents has completely reference homozygotes and another has laternative homozygotes at all markers, Set dummy_reads = 0 to leave uncertainty to estimate parental genotypes based on the offspring genotypes. Nevertheless, the parentless mode is less accurate and has more chance to get a genotype estimate randomly selected from the equally likely genotype estimates.

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", getSamID(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)

# 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, 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)
```

GBScleanR *GBScleanR: A package to conduct error correction for noisy genotyping by sequencing (reduced representation sequencing based genotyoing) data.*

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 "clean.geno" 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. GBScleanR currenly only supports genotype data of biparental populations.

Author(s)

Maintainer: Tomoyuki Furuta <f.tomoyuki@okayama-u.ac.jp> (ORCID)

See Also

Useful links:

- https://github.com/tomoyukif/GBScleanR
- Report bugs at https://github.com/tomoyukif/GBScleanR/issues

---

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", "hap", "dosage to output raw genotype data, filtered genotype data, corrected genotype data, estimated haplotype data, and estimated allele dosage 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 setting `format = "qtl"` makes the function export the data in the r/qtl format that can be loaded using read.cross as `format = "csvs"` with a phenotype data. If you have executed `estGeno()` and your population is a biparental population, set 'node = "dosage"' to export a r/qtl format CSV in which homozygoutes of the alleles of Parent 1 and 2, which have been specified by `setParents()`, are represented by A and B, respectively. If 'node = "raw"', 'node = "fill"', and 'node = "cor"', A and B in the r/qtl format CSV indicate homozygoutes of reference and alternative alleles shown in a given VCF file. This means that if Parent 1 has the alternative allele homozygoute at Marker 1 and Offspring 1 has the same genotype with Parent 1, the genotype of Offspring 1 at Marker 1 will be B in the r/qtl format CSV. On the other hand, if you set 'node = "dosage"', the genotype of Offspring 1 at Marker 1 will be A in the r/qtl format CSV. The output CSV file has 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",
  info.export = NULL,
  fmt.export = NULL,
```

```
  parents = TRUE,
  ...
)

## S4 method for signature 'GbsrGenotypeData'
gbsrGDS2VCF(object, out_fn, node, info.export, fmt.export, parents)
```

Arguments

object	A GbsrGenotypeData object.
out_fn	A string to specify the path to an output VCF file.
node	Either one of "raw" or "cor" to output raw genotype data or corrected genotype data, respectively.
info.export	characters, the variable name(s) in the INFO node for export; or NULL for all variables. If you specify `character(0)`, nothing will be exported from the INFO node.
fmt.export	characters, the variable name(s) in the FORMAT node for import; or NULL for all variables. If you specify `character(0)`, nothing will be exported from the FORMAT node, except for GT.
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 `reopenGDS()` 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)

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

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 is an extention of `SeqVarGDSClass-class` in the `SeqArray` package to store summary data of genotypes and reads and a GbsrScheme object that contains mating scheme information of the given population.. The slots `marker` and `sample` store a data.frame object for variant-wise and sample-wise summary information, respectively. The `scheme` slot holds a GbsrScheme object. The function `loadGDS()` initialize the GbsrGenotypeData class.

Slots

`marker` A data.frame object.

`sample` A data.frame object.

`scheme` A GbsrScheme object.

Examples

```
# [loadGDS()] initialize the [GbsrGenotypeData] object.

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

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

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.

`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.

`samples` A vector of member IDs of samples indicating which samples are derived from which pedigrees.

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")
gds <- loadGDS(gds_fn)

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

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

`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,
  gt = "GT",
  info.import = NULL,
  fmt.import = NULL,
  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.
gt	the ID for genotypic data in the FORMAT column; "GT" by default, VCFv4.0.
info.import	characters, the variable name(s) in the INFO field for import; or NULL for all variables. If you specify character(0), nothing will be retrieved from the INFO filed.
fmt.import	characters, the variable name(s) in the FORMAT field for import; or NULL for all variables. If you specify character(0), nothing will be retrieved from the FORMAT filed, except for GT.
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 seqVCF2GDS of SeqArray.

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.
gds <- loadGDS(gds_fn)

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

---

getAllele *Obtain reference allele information of markers*

Description

This function returns the reference allele and alternative allele(s).

Usage

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

## S4 method for signature 'GbsrGenotypeData'
getAllele(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 alleles of markers which are labeled TRUE in the "valid" column of the "marker" slot will be returned. If you need the number of over all markers, set valid = FALSE. validMar() tells you which markers are valid.

Value

A vector of strings each of which is a "/" separated string and indicates the reference allele and the alternative allele(s) at a 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)

getAllele(gds)

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

---

getChromosome *Obtain chromosome IDs of markers*

Description

This function returns chromosome IDs of markers.

Usage

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

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

Arguments

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

Details

If valid = TRUE, the chromosome IDs of the markers which are labeled TRUE in the "valid" column of the "marker" slot will be returned. If you need the number of over all markers, set valid = FALSE. validMar() tells you which markers are valid.

Value

A vector of factors 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 sample*

Description

Obtain total alternative allele counts per SNP or per sample

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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 sample*

Description

Obtain total missing allele counts per SNP or per sample

Usage

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

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

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "marker" and "sample".
`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 sample slot will be returned. `validMar()` 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
# [marker] and [sample] 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 sample*

Description

Obtain total reference allele counts per SNP or per sample

Usage

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

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

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "marker" and "sample".
`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 sample slot will be returned. `validMar()` 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
# [marker] and [sample] 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 sample*

Description

Obtain total alternative genotype counts per SNP or per sample

Usage

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

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

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "marker" and "sample".
`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 sample slot will be returned. `validMar()` 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
# [marker] and [sample] 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 sample*

Description

Obtain total heterozygote counts per SNP or per sample

Usage

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

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

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "marker" and "sample".
`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 sample slot will be returned. `validMar()` 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
# [marker] and [sample] 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 sample*

Description

Obtain total missing genotype counts per SNP or per sample

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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 sample*

Description

Obtain total reference genotype counts per SNP or per sample

Usage

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

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

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "marker" and "sample".
`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 sample slot will be returned. `validMar()` 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
# [marker] and [sample] 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 sample*

Description

Obtain total read counts per SNP or per sample

Usage

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

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

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "marker" and "sample".
`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 sample slot will be returned. `validMar()` 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)
gds <- countRead(gds)

getCountRead(gds)

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

getCountReadAlt *Obtain total alternative read counts per SNP or per sample*

Description

Obtain total alternative read counts per SNP or per sample

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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)
gds <- countRead(gds)

getCountReadAlt(gds)

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

getCountReadRef *Obtain total reference read counts per SNP or per sample*

Description

Obtain total reference read counts per SNP or per sample

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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)
gds <- countRead(gds)

getCountReadRef(gds)

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

getFixedParameter *Get fixed allele read biases and mismapping rate*

Description

Get fixed allele read biases and mismapping rates of markers

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
chr	A integer or string to specify chromosome to get information.
...	Unused.

Details

If valid = TRUE, A logical vector for the markers which are labeled TRUE in the "valid" column of the "marker" slot will be returned. If you need check the dominant markers in all markers, set valid = FALSE. validMar() tells you which markers are valid.

Value

A numeric vector of fixed allele read biases.

A GbsrGenotypeData object after adding dominant marker information

See Also

setFixedParameter()

Examples

```
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)

# Not run.
# Run estGeno() and reuse the estimated parameters in the second run.
# gds <- makeScheme(gds, generation = 2, crosstype = "self")
# gds <- estGeno(gds)
# fixed_param <- getFixedParameter(gds)
# gds <- setFixedParameter(gds,
#                          bias = fixed_param$bias,
```

```
#                           mismap = fixed_param$mismap)
# gds <- estGeno(gds, optim = FALSE, call_threshold = 0.5)

# 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,
  node = "raw",
  parents = FALSE,
  valid = TRUE,
  chr = NULL,
  phased = FALSE,
  ...
)

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

Arguments

`object`	A GbsrGenotypeData object.
`node`	Either of "raw", "filt", "cor", and "parents". See details.
`parents`	A logical value or "only" whether to include data for parents or not or to get data only for parents. Ignored if `node = "parents`.
`valid`	A logical value. See details.
`chr`	A integer vector of indexes indicating chromosomes to get read count data.
`phased`	If set `TRUE` to `phased`, the function will return phased genotype data in a P x N x M array where P, N, and M are the ploidy, number of samples, and number of markers.
`...`	Unused.

Details

When `node = "raw`, the raw genotype data stored in the "genotype/data" node will be returned, while `node = "filt` make the function to return the filtered genotype data stored in the "annotation/format/FGT/data" that can be generated via the `setCallFilter()` function. `node = "cor` indicates to get the corrected genotype data stored in the "annotation/format/CGT/data" that can be generated via the `estGeno()` function. The estimated parental genotypes, which also can be generated via the `estGeno()` function and stored in the "annotation/info/PGT" node, can be obtained with `node = "parents"`. If `valid = TRUE`, genotype calls for only valid marker and valid samples will be obtained.

Value

An integer matrix of genotype data which is represented by the number of alternative 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, parents = FALSE, valid = TRUE, chr = NULL, ...)

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

Arguments

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

# 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, 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 specify chromosome to get information.
...	Unused.

Details

If valid = TRUE, the information of the markers which are labeled TRUE in the "valid" column of the "marker" slot will be returned. If you need the number of over all markers, set valid = FALSE. validMar() tells you which markers 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 sample*

Description

Obtain minor allele counts per SNP or per sample

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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
# [marker] and [sample] 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 sample*

Description

Obtain minor allele frequencies per SNP or per sample

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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
# [marker] and [sample] slots of the [GbsrGenotypeData] object.
gds <- countGenotype(gds)

getMAF(gds)

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

getMarID *Obtain the marker IDs*

Description

Obtain the marker IDs

Usage

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

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

Arguments

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

Details

If valid = TRUE, the IDs of markers which are labeled TRUE in the "valid" column of the "marker" slot will be returned. If you need the number of over all markers, set valid = FALSE. validMar() tells you which markers are valid.

Value

A integer vector of marker 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)

getMarID(gds)

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

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

Description

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

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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 [marker] and [sample] slots
# of the [GbsrGenotypeData] object.
gds <- countRead(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 sample*

Description

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

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
valid	A logical value. See details.
...	Unused.

Details

You need to execute countRead() to calculate summary statistics to be obtained via this function. If valid = TRUE, the chromosome information of markers which are labeled TRUE in the sample slot will be returned. validMar() 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 <- countRead(gds)
getMeanReadRef(gds)
closeGDS(gds) # Close the connection to the GDS file
```

`getMedianReadAlt` *Obtain quantile values of total alternative read counts per SNP or per sample*

Description

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

Usage

```
getMedianReadAlt(object, target = "marker", valid = TRUE, ...)

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

Arguments

`object`	A GbsrGenotypeData object.
`target`	Either of "marker" and "sample".
`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 sample slot will be returned. `validMar()` 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 [marker] and [sample] slots
# of the [GbsrGenotypeData] object.
gds <- countRead(gds)

getMedianReadAlt(gds)

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

getMedianReadRef *Obtain quantile values of total reference read counts per SNP or per sample*

Description

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

Usage

```
getMedianReadRef(object, target = "marker", valid = TRUE, ...)

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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 [marker] and [sample] slots
# of the [GbsrGenotypeData] object.
gds <- countRead(gds)

getMedianReadRef(gds)

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

getParents *Get parental sample information*

Description

This function returns sample IDs, member IDs and indexes of parental samples set via `setParents()`. Sample 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, verbose = TRUE, ...)

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

Arguments

`object`	A GbsrGenotypeData object.
`bool`	If TRUE, the function returns a logical vector indicating which samples have been set as parents.
`verbose`	If FALSE, the function does not print a warning message even when parents were not specified in the given GbsrGenotypeData object. The setting `verbose = FALSE` is used in the other functions to call `getParents()` without evoking unnecessary warnings to users.
`...`	Unused.

Value

A data frame of parents information indicating sampleIDs, 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", getSamID(gds), value = TRUE)

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

# Get the information of parents.
getParents(gds)

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

getPloidy *Get ploidy*

Description

Get the ploidy of a given population in the input GbsrGenotypeData object.

Usage

```
getPloidy(object, ...)

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

Arguments

object	A GbsrGenotypeData object.
...	Unused.

Value

An integer value.

See Also

setPloidy()

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)

ploidy <- getPloidy(gds)
print(ploidy)

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

getPosition *Obtain marker positions*

Description

This function returns physical positions of 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 integer or string to specify chromosome to get information.
...	Unused.

Details

If valid = TRUE, the positions of the markers which are labeled TRUE in the "valid" column of the "marker" slot will be returned. If you need the positions of over all markers, set valid = FALSE. validMar() tells you which markers are valid.

Value

An integer vector 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)
```

---

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, node = "raw", parents = FALSE, valid = TRUE, chr = NULL, ...)

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

Arguments

object	A GbsrGenotypeData object.
node	Either of "raw" and "filt". See details.
parents	A logical value or "only" whether to include data for parents or not or to get data only for parents.
valid	A logical value. See details.
chr	An integer vector of indexes indicating chromosomes to get read count data.
...	Unused.

Details

When `node = "raw"`, the raw read counts stored in the "annotation/format/AD/data" node will be returned, while `node = "filt"` make the function to return the filtered read counts stored in the "annotation/format/FAD/data" that can be generated via the `setCallFilter()` function. 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)
```

---

getReplicates *Get identifiers to indicates which samples are replicates.*

Description

Not implemented yet. This function assign identifiers that indicates which samples are replicates those which should have the same genotypes at all markers.

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
parents	A logical value to indicate whether to include replicate IDs for parental samples in the output. If you specify parents = "only", this function returns replicate IDs only for parental samples.
...	Unused.

Details

The replicates of samples specified in setReplicates() will have the same genotypes at all markers in the estimated genotypes obtained via estGeno(). In the genotype estimation by estGeno(), the Viterbi scores for each possible genotype (haplotype) at each marker for the replicates will be replaced with the average score for the replicates.

Value

A GbsrGenotypeData object with genotype count information.

See Also

setReplicates()

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 <- setParents(gds, parents = c("Founder1", "Founder2"))

# When your data has 100 samples, two replicates for each offspring,
# and the samples are ordered as the 1st replicate followed by the 2nd
# replicate, you can specify replicates as below.
# gds <- setReplicates(gds, replicates = rep(1:50, each = 2))

# If you need to confirm the order of samples, run the following code.
# id <- getSamID(gds)

# Replicate IDs should be set also to parents. Therefore, please include


# getReplicates(gds)

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

getSamID *Obtain the sample IDs*

Description

This function returns sample IDs.

Usage

```
getSamID(object, valid = TRUE, parents = FALSE, ...)

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

Arguments

object	A GbsrGenotypeData object.
valid	A logical value. See details.
parents	A logical value whether to include data for parents or not.
...	Unused.

Details

If valid = TRUE, the IDs of samples which are labeled TRUE in the "valid" column of the "sample" slot will be returned. If you need the number of over all samples, set valid = FALSE. validSam() tells you which samples are valid.

Value

A character vector of 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)

getSamID(gds)

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

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

Description

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

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
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 sample slot will be returned. validMar() 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 [marker] and [sample] slots
# of the [GbsrGenotypeData] object.
gds <- countRead(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 sample*

Description

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

Usage

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

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

Arguments

object	A GbsrGenotypeData object.
target	Either of "marker" and "sample".
valid	A logical value. See details.
...	Unused.

Details

You need to execute countRead() to calculate summary statistics to be obtained via this function. If valid = TRUE, the chromosome information of markers which are labeled TRUE in the sample slot will be returned. validMar() 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 [marker] and [sample] slots
# of the [GbsrGenotypeData] object.
gds <- countRead(gds)

getSDReadRef(gds)

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

`histGBSR` *Draw histograms of specified statistics*

Description

Draw histograms of specified statistics

Usage

```
histGBSR(
  x,
  stats = c("dp", "missing", "het"),
  target = c("marker", "sample"),
  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 "marker" and "sample", e.g. `target = "marker"` to draw a histogram only for SNPs.
`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.

median_ref Quantile of reference allele read counts.

mean_alt Mean of alternative allele read counts.

sd_alt Standard deviation of alternative allele read counts.

median_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()`. "mq", "fs", "qd", "sor", "mqranksum", "readposranksum", and "baseqranksum" only work with `target = "marker"`, 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")
gds <- loadGDS(gds_fn)

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

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

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

---

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, mating, ...)

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

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

Arguments

object	A GbsrGenotypeData object.
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, 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, load_filter = FALSE, ploidy = 2, verbose = TRUE)
```

Arguments

x	A string of the path to an input GDS file or a GbsrGenotypeData object to reload.
load_filter	A logical whether to load the filtering information made via `setSamFilter()` and `setMarFilter()` and saved in the GDS file via `closeGDS()` with `save_filter = TRUE`.
ploidy	Set the ploidy of the population.
verbose	if TRUE, show information.

Details

The first time to load a newly produced GDS file will take long time due to data reformatting for quick access. The GbsrGenotypeData object returned from `loadGDS()` can be also handled as `SeqVarGDSClass-class` of the SeqArray package.

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)

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

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

makeScheme *Automate a GbsrScheme object building.*

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 automates the building of a GbsrScheme object.

Usage

```
makeScheme(object, generation, crosstype, ...)

## S4 method for signature 'GbsrGenotypeData'
makeScheme(object, generation, crosstype)
```

Arguments

`object`	A GbsrGenotypeData object.
`generation`	An integer to indicate which generation of selfing or sibling-crossing your population is.
`crosstype`	A string to indicate the type of cross conducted with 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` specified to `makeScheme()` can take "selfing" and "sibling". When your population has 2^n parents specified by `setParents()`, `makeScheme()` assumes those parents were crossed in the "funnel" design in which 2^n parents are crossed to obtain $2^n/2$ F1 hybrids followed by successive intercrossings (pairings) of the hybrids to combine the genomes of all parents in one family of siblings. The `makeScheme()` function assumes that the parents that were assigned an odd number member ID (N) in `setParents()` had been crossed with the parent that were assigned an even number (N+1). For example, if you set parents as shown below. The `makeScheme()` function prepare a scheme information that indicates the intercrossings of "p1 x p2", "p3 x p4", "p5 x p6", and "p7 x p8" followed by crossing of "p1xp2_F1 x p3xp4_F1" and "p5xp6_F1 x p7xp8_F1" and then crossing of the two 4-way crossed liens to produce 8-way crossed hybrid lines. If, for example, `generation = 5` indicating an F5 generation was specified to `makeScheme()`, the function adds 4 successive selfing or sibling crossings in the scheme. The created GbsrScheme object will be set in the `scheme` slot of the GbsrGenotypeData object.

Value

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

See Also

`initScheme()`, `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"))

gds <- makeScheme(gds, generation = 2, crosstype = "self")

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

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

`nmar` *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

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

## S4 method for signature 'GbsrGenotypeData'
nmar(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 markers which are labeled `TRUE` in the "valid" column of the "marker" slot will be returned. If you need the number of over all markers, set `valid = FALSE`. `validMar()` tells you which markers are valid.

Value

An integer value to indicate the number of SNP markers.

See Also

`validMar()`

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)

nmar(gds)

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

`nsam` *Return the number of samples.*

Description

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

Usage

```
nsam(object, valid = TRUE, parents = FALSE, ...)

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

Arguments

`object`	A GbsrGenotypeData object.
`valid`	A logical value. See details.
`parents`	A logical value whether to include to parental samples or not.
`...`	Unused.

Details

If `valid = TRUE`, the number of the samples which are labeled `TRUE` in the "valid" column of the "sample" slot will be returned. If you need the number of over all samples, set `valid = FALSE`. `validSam()` tells you which samples are valid.

Value

An integer value to indicate the number of samples.

See Also

`validSam()`

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)

nsam(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 = "marker",
  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 "marker" and "sample", e.g. `target = "marker"` to draw a histogram only for SNPs.
`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.

median_ref Quantile of reference allele read counts.

mean_alt Mean of alternative allele read counts.

sd_alt Standard deviation of alternative allele read counts.

median_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()`. "mq", "fs", "qd", "sor", "mqranksum", "readposranksum", and "baseqranksum" only work with `target = "marker"`, 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")
gds <- loadGDS(gds_fn)

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

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

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

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",
  showratio = TRUE,
  dot_fill = c("green", "darkblue"),
  size = 0.8,
  alpha = 0.8,
  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" or "filt" to output raw read data, or filtered read data, respectively.
showratio	If TRUE, draw dots indicating read ratio.
dot_fill	A vector of two strings to indicate the dot colors in the plot. The first and second elements of the vector are set as the colors for the lowest and highest values in the gradient coloring of the dots indicating total read counts par marker.
size	A positive number to indicate the dot size in a plot.
alpha	A positive number in 0-1 to indicate the dot opacity in a 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")
gds <- loadGDS(gds_fn)

plotDosage(gds, ind = 1)

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

`plotGBSR` *Draw line plots of specified statistics*

Description

Draw line plots of specified statistics

Usage

```
plotGBSR(
  x,
  stats = c("dp", "missing", "het"),
  coord = NULL,
  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.
`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.

median_ref Quantile of reference allele read counts.

mean_alt Mean of alternative allele read counts.

sd_alt Standard deviation of alternative allele read counts.

median_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()`. "mq", "fs", "qd", "sor", "mqranksum", "readposranksum", #' and "baseqranksum" only work with `target = "marker"`, 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")
gds <- loadGDS(gds_fn)

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

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

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

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 = c("green", "darkblue"),
  size = 0.8,
  alpha = 0.8
)
```

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	A string of sample id or an index to specify the sample to be drawn.
node	Either one of "raw" or "filt" to output raw read data, or filtered read data, respectively.
dot_fill	A vector of two strings to indicate the dot colors in the plot. The first and second elements of the vector are set as the colors for the lowest and highest values in the gradient coloring of the dots indicating total read counts par marker.
size	A positive number to indicate the dot size in the plot.
alpha	A positive number in 0-1 to indicate the dot opacity 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")
gds <- loadGDS(gds_fn)

plotReadRatio(gds, ind = 1)

# Close the connection to the GDS file
```

```
closeGDS(gds)
```

`reopenGDS` *Reopen the connection to the GDS file.*

Description

Reopen the connection to the GDS file.

Usage

```
reopenGDS(object, ...)

## S4 method for signature 'GbsrGenotypeData'
reopenGDS(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 marker and sample slots 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 <- reopenGDS(gds)

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

`resetCallFilter` *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

```
resetCallFilter(object, ...)

## S4 method for signature 'GbsrGenotypeData'
resetCallFilter(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. `resetCallFilter()` set back the nodes to the original, those are "genotype" and "data" for genotype and read count data, respectively.

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 <- resetCallFilter(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)
```

`resetFilter` *Reset all filters made by* `setSamFilter()`, `setMarFilter()`, *and* `setCallFilter()`.

Description

Return all data intact.

Usage

```
resetFilter(object, ...)

## S4 method for signature 'GbsrGenotypeData'
resetFilter(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))

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

gds <- setSamFilter(gds,
```

```
                        id = getSamID(gds)[1:10],
                        missing = 0.2,
                        dp = c(5, Inf))

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

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

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

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

`resetMarFilter` *Reset the filter made by* `setMarFilter()`

Description

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

Usage

```
resetMarFilter(object, ...)

## S4 method for signature 'GbsrGenotypeData'
resetMarFilter(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.
nmar(gds)

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

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

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

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

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

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

`resetSamFilter` *Reset the filter made by* `setSamFilter()`

Description

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

Usage

```
resetSamFilter(object, ...)

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

Arguments

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

Value

A GbsrGenotypeData object after removing all filters on 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 <- setSamFilter(gds,
                        id = getSamID(gds)[1:10],
                        missing = 0.2,
                        dp = c(5, Inf))

# Reset all filters applied above.
gds <- resetSamFilter(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),
  dp_qtile = c(0, 1),
  ref_qtile = c(0, 1),
  alt_qtile = c(0, 1),
  ...
)

## S4 method for signature 'GbsrGenotypeData'
setCallFilter(
  object,
  dp_count,
  ref_count,
  alt_count,
  dp_qtile,
  ref_qtile,
  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.
dp_qtile	A numeric vector with length two specifying lower and upper limit of quantile of total read counts in each sample.
ref_qtile	A numeric vector with length two specifying lower and upper limit of quantile of reference read counts in each sample.
alt_qtile	A numeric vector with length two specifying lower and upper limit of quantile of alternative read counts in each sample.
...	Unused.

Details

dp_qtile, ref_qtile, and alt_qtile 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 filtered read counts and genotype calls will be stored in the data node in the "FAD" folder and the data node in the "FGT" folder, while the data node in the "CFT" stores call fitering informatin. To reset the filter applied by setCallFilter(), run resetCallFilter().

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, dp_qtile = c(0.2, 1))

# Reset the filter
gds <- resetCallFilter(gds)

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

`setFixedParameter` *Set fixed allele read biases and mismapping rate*

Description

Set fixed allele read biases and mismapping rates of markers

Usage

```
setFixedParameter(object, bias = NULL, mismap = NULL, parent_geno = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
setFixedParameter(object, bias, mismap, parent_geno)
```

Arguments

`object`	A GbsrGenotypeData object.
`bias`	A numeric vector of fixed allele read biases to be assigned to valid markers. The length of `bias` vector should match the number of valid markers.
`mismap`	A numeric matrix of fixed reference and alternative read mismapping rates to be assigned to valid markers. The number of rows of the given matrix should match the number of valid markers and should have two columns that are for reference and alternative read mismapping rates, respectively.
`parent_geno`	A logical value indicating whether to use fixed parental genotypes in the genoype estimation by `estGeno()`. This mode requires the estimated genotypes for parental samples that were estimated by `estGeno()` and stored in the GDS file linked to the input GbsrGenotypeData object.
`...`	Unused.

Details

If you have already executed genotype estimation and want to reuse the marker-wise allele read biases and mismapping rates estimated in the completed run of estGeno(), you can use them in the next genotype estimation run. For example, if you want to estimate genotypes with different argument settings of setGeno(), it is worth to set fixed parameters and run estGeno() with setting `optim = FALSE` to skip time-consuming iterative parameter optimization steps but use the estimated parameters from the first run to incorporate the marker-wise error parameters. Since the bias set by `setFixedParameter()` function is the reference allele read bias, the values 0 and 1 mean that the marker only gives alternative and reference allele reads, respectively. The values in the `bias` vector are assigned to the valid markers. Similarly, the values in the `mismap` matrix are assigned to the valid markers in the order they appear in the rows.

Value

A GbsrGenotypeData object after adding dominant marker information

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)

# Not run.
# Run estGeno() and reuse the estimated parameters in the second run.
# gds <- makeScheme(gds, generation = 2, crosstype = "self")
# gds <- estGeno(gds)
# fixed_param <- getFixedParameter(gds)
# gds <- setFixedParameter(gds,
#                          bias = fixed_param$bias,
#                          mismap = fixed_param$mismap)
# gds <- estGeno(gds, optim = FALSE, call_threshold = 0.5)

# You can also set arbitrary values.
bias <- sample(seq(0, 1, 0.01), nmar(gds), replace = TRUE)
mismap <- cbind(sample(seq(0, 0.2, 0.01), nmar(gds), replace = TRUE),
                sample(seq(0, 0.2, 0.01), nmar(gds), replace = TRUE))
gds <- setFixedParameter(gds, bias = bias, mismap = mismap)

# 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)
```

---

setMarFilter *Filter out markers*

Description

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

Usage

```
setMarFilter(
  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'
setMarFilter(
  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 getMarID(). 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 validMar().

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

# 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, nonmiss = FALSE, mono = FALSE, bi = FALSE, ...)

## S4 method for signature 'GbsrGenotypeData'
setParents(object, parents, nonmiss, 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 getSamID().
nonmiss	A logical value whether to filter out markers which are missing in parents.
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. The replicates of parental samples specified to the repliate argument of setParents() will have the same genotypes at all markers in the estimated genotypes obtained via estGeno(). In the genotype estimation by estGeno(), the Viterbi scores for each possible genotype at each marker for the replicates will be replaced with the average score for the replicates.

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", getSamID(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)

# 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, 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)
```

`setPloidy` *Set ploidy*

Description

Set the ploidy of a given population in the input GbsrGenotypeData object.

Usage

```
setPloidy(object, ploidy = 2, ...)

## S4 method for signature 'GbsrGenotypeData'
setPloidy(object, ploidy = 2)
```

Arguments

`object`	A GbsrGenotypeData object.
`ploidy`	A integer value to specify the ploidy of the given population.
`...`	Unused.

Details

The genotype estimation by `estGeno()` would be performed with the assumption of the ploidy specified through this function or the `ploidy` argument of `loadGDS()`. When an odd number was specified as `ploidy`, the ploidy of intermediate generations would be treated as `ploidy + 1` to properly list up possible descendent haplotype patterns in the process by `estGeno()`.

Value

A GbsrGenotypeData object with filters on markers.

See Also

`getPloidy()`

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 <- setPloidy(gds, ploidy = 4)

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

---

setReplicates *Set identifiers to indicates which samples are replicates.*

Description

Not implemented yet. This function assign identifiers that indicates which samples are replicates those which should have the same genotypes at all markers.

Usage

```
setReplicates(object, replicates, ...)

## S4 method for signature 'GbsrGenotypeData'
setReplicates(object, replicates)
```

Arguments

`object`	A GbsrGenotypeData object.
`replicates`	A vector of integers, numbers, or characters to indicate grouping of samples as replicates.
`...`	Unused.

Details

The replicates of samples specified in `setReplicates()` will have the same genotypes at all markers in the estimated genotypes obtained via `estGeno()`. In the genotype estimation by `estGeno()`, the Viterbi scores for each possible genotype (haplotype) at each marker for the replicates will be replaced with the average score for the replicates.

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)

gds <- setParents(gds, parents = c("Founder1", "Founder2"))
```

```
# When your data has 100 samples, two replicates for each offspring,
# and the samples are ordered as the 1st replicate followed by the 2nd
# replicate, you can specify replicates as below.
# gds <- setReplicates(gds, replicates = rep(1:50, each = 2))

# If you need to confirm the order of samples, run the following code.
# id <- getSamID(gds)

# Replicate IDs should be set also to parents. Therefore, please include

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

`setSamFilter` *Filter out samples*

Description

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

Usage

```
setSamFilter(
  object,
  id = NA_character_,
  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'
setSamFilter(
  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 sample ID which can be retrieve by getSamID(). 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 validSam().

Value

A GbsrGenotypeData object with filters on 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 <- setSamFilter(gds,
                        id = getSamID(gds)[1:10],
                        missing = 0.2,
                        dp = c(5, Inf))

# 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, pedigree)
```

Arguments

`object`	A GbsrGenotypeData object.
`...`	Unused.
`parents_name`	A vector of strings to indicate names of parental samples. This argument is used internally by showScheme() for the gbsrGenotypeData object.
`pedigree`	A integer vector indicating the member ID assignment to samples. This argument is used internally by showScheme() for the gbsrGenotypeData object.

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, 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)
```

---

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 `validMar()` 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
```

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

Description

Return a logical vector indicating which are valid SNP markers.

Usage

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

validMar(object) <- value

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

## S4 replacement method for signature 'GbsrGenotypeData'
validMar(object) <- value
```

Arguments

`object`	A GbsrGenotypeData object.
`chr`	A index to spefcify chromosome to get information.
`...`	Unused.
`value`	A logical vector indicating valid markers with the length matching with the number of markers.

Value

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

validMar(gds)

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

`validSam` *Return a logical vector indicating which are valid samples.*

Description

Return a logical vector indicating which are valid samples.

Usage

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

validSam(object) <- value

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

## S4 replacement method for signature 'GbsrGenotypeData'
validSam(object) <- value
```

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 indicating valid samples with the length matching with the number of samples

Value

A logical vector of the same length with the number of total 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)

validSam(gds)

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

Index

∗ **internal**
GBScleanR, 13

addScheme, 3
addScheme(), *4, 5*, *53*, *56*, *83*
addScheme,GbsrGenotypeData-method *(addScheme)*, 3
addScheme,GbsrScheme-method *(addScheme)*, 3
assignScheme, 5
assignScheme(), *5*
assignScheme,GbsrGenotypeData-method *(assignScheme)*, 5
assignScheme,GbsrScheme-method *(assignScheme)*, 5

boxplotGBSR, 6

closeGDS, 8
closeGDS(), *55*
closeGDS,GbsrGenotypeData-method *(closeGDS)*, 8
countGenotype, 9
countGenotype(), *7*, *21–27*, *36*, *37*, *52*, *60*, *63*
countGenotype,GbsrGenotypeData-method *(countGenotype)*, 9
countRead, 10
countRead(), *7*, *28–30*, *39–42*, *49*, *50*, *52*, *60*, *63*
countRead,GbsrGenotypeData-method *(countRead)*, 10

data.frame, *17*

estGeno, 11
estGeno(), *4*, *5*, *9*, *12*, *15*, *17*, *32–34*, *47*, *52*, *53*, *56*, *72*, *77–79*
estGeno,GbsrGenotypeData-method *(estGeno)*, 11

GBScleanR, *3*, *5*, 13, *17*, *52*, *56*, *66*, *73*, *77*
GBScleanR-package *(GBScleanR)*, 13
gbsrGDS2CSV, 14
gbsrGDS2CSV,GbsrGenotypeData-method *(gbsrGDS2CSV)*, 14
gbsrGDS2VCF, 15
gbsrGDS2VCF,GbsrGenotypeData-method *(gbsrGDS2VCF)*, 15
GbsrGenotypeData, *4*, *5*, *7–10*, *12*, *14*, *16–18*, *20–33*, *35–51*, *53–59*, *61*, *62*, *64–72*, *74–79*, *81–85*
GbsrGenotypeData *(GbsrGenotypeData-class)*, 17
GbsrGenotypeData-class, 17
GbsrScheme, *3–5*, *17*, *52*, *53*, *56*, *82*
GbsrScheme *(GbsrScheme-class)*, 17
GbsrScheme-class, 17
gbsrVCF2GDS, 18
getAllele, 19
getAllele,GbsrGenotypeData-method *(getAllele)*, 19
getChromosome, 20
getChromosome,GbsrGenotypeData-method *(getChromosome)*, 20
getCountAlleleAlt, 21
getCountAlleleAlt,GbsrGenotypeData-method *(getCountAlleleAlt)*, 21
getCountAlleleMissing, 22
getCountAlleleMissing,GbsrGenotypeData-method *(getCountAlleleMissing)*, 22
getCountAlleleRef, 23
getCountAlleleRef(), *9*
getCountAlleleRef,GbsrGenotypeData-method *(getCountAlleleRef)*, 23
getCountGenoAlt, 24
getCountGenoAlt,GbsrGenotypeData-method *(getCountGenoAlt)*, 24
getCountGenoHet, 25
getCountGenoHet,GbsrGenotypeData-method *(getCountGenoHet)*, 25
getCountGenoMissing, 26
getCountGenoMissing,GbsrGenotypeData-method *(getCountGenoMissing)*, 26
getCountGenoRef, 27
getCountGenoRef(), *9*
getCountGenoRef,GbsrGenotypeData-method *(getCountGenoRef)*, 27
getCountRead, 28

getCountRead,GbsrGenotypeData-method
(getCountRead), 28
getCountReadAlt, 29
getCountReadAlt(), *10*
getCountReadAlt,GbsrGenotypeData-method
(getCountReadAlt), 29
getCountReadRef, 30
getCountReadRef(), *10*
getCountReadRef,GbsrGenotypeData-method
(getCountReadRef), 30
getFixedParameter, 31
getFixedParameter,GbsrGenotypeData-method
(getFixedParameter), 31
getGenotype, 32
getGenotype,GbsrGenotypeData-method
(getGenotype), 32
getHaplotype, 33
getHaplotype,GbsrGenotypeData-method
(getHaplotype), 33
getInfo, 35
getInfo,GbsrGenotypeData-method
(getInfo), 35
getMAC, 36
getMAC,GbsrGenotypeData-method
(getMAC), 36
getMAF, 37
getMAF(), *9*
getMAF,GbsrGenotypeData-method
(getMAF), 37
getMarID, 38
getMarID,GbsrGenotypeData-method
(getMarID), 38
getMeanReadAlt, 39
getMeanReadAlt,GbsrGenotypeData-method
(getMeanReadAlt), 39
getMeanReadRef, 40
getMeanReadRef(), *10*
getMeanReadRef,GbsrGenotypeData-method
(getMeanReadRef), 40
getMedianReadAlt, 41
getMedianReadAlt(), *10*
getMedianReadAlt,GbsrGenotypeData-method
(getMedianReadAlt), 41
getMedianReadRef, 42
getMedianReadRef,GbsrGenotypeData-method
(getMedianReadRef), 42
getParents, 43
getParents,GbsrGenotypeData-method
(getParents), 43
getPloidy, 44
getPloidy(), *78*
getPloidy,GbsrGenotypeData-method
(getPloidy), 44
getPosition, 44
getPosition,GbsrGenotypeData-method
(getPosition), 44
getRead, 45
getRead,GbsrGenotypeData-method
(getRead), 45
getReplicates, 46
getReplicates,GbsrGenotypeData-method
(getReplicates), 46
getSamID, 48
getSamID(), *77*
getSamID,GbsrGenotypeData-method
(getSamID), 48
getSDReadAlt, 49
getSDReadAlt,GbsrGenotypeData-method
(getSDReadAlt), 49
getSDReadRef, 50
getSDReadRef,GbsrGenotypeData-method
(getSDReadRef), 50
ggplot2::stat_smooth(), *59*

histGBSR, 51

initScheme, 52
initScheme(), *4*, *12*, *56*, *83*
initScheme,GbsrGenotypeData-method
(initScheme), 52
initScheme,GbsrScheme-method
(initScheme), 52
isOpenGDS, 54
isOpenGDS,GbsrGenotypeData-method
(isOpenGDS), 54

loadGDS, 55
loadGDS(), *8*, *16–18*, *55*, *78*

makeScheme, 56
makeScheme(), *56*
makeScheme,GbsrGenotypeData-method
(makeScheme), 56

nmar, 57
nmar,GbsrGenotypeData-method (nmar), 57
nsam, 58
nsam,GbsrGenotypeData-method (nsam), 58

pairsGBSR, 59
plotDosage, 61
plotGBSR, 62
plotReadRatio, 64

reopenGDS, 65
reopenGDS(), *16*

reopenGDS,GbsrGenotypeData-method
(reopenGDS), 65
resetCallFilter, 66
resetCallFilter(), 66, 71
resetCallFilter,GbsrGenotypeData-method
(resetCallFilter), 66
resetFilter, 67
resetFilter,GbsrGenotypeData-method
(resetFilter), 67
resetMarFilter, 68
resetMarFilter,GbsrGenotypeData-method
(resetMarFilter), 68
resetSamFilter, 69
resetSamFilter,GbsrGenotypeData-method
(resetSamFilter), 69

sample, 9, 10, 21–30, 36, 37, 39–42, 49, 50
seqVCF2GDS, 19
setCallFilter, 70
setCallFilter(), 9, 10, 12, 32, 33, 46, 66, 67
setCallFilter,GbsrGenotypeData-method
(setCallFilter), 70
setFixedParameter, 72
setFixedParameter(), 31, 72
setFixedParameter,GbsrGenotypeData-method
(setFixedParameter), 72
setInfoFilter, 73
setInfoFilter,GbsrGenotypeData-method
(setInfoFilter), 73
setMarFilter, 74
setMarFilter(), 8, 55, 67, 68
setMarFilter,GbsrGenotypeData-method
(setMarFilter), 74
setParents, 76
setParents(), 12, 15, 43, 53, 56, 77
setParents,GbsrGenotypeData-method
(setParents), 76
setPloidy, 78
setPloidy(), 44
setPloidy,GbsrGenotypeData-method
(setPloidy), 78
setReplicates, 79
setReplicates(), 47, 79
setReplicates,GbsrGenotypeData-method
(setReplicates), 79
setSamFilter, 80
setSamFilter(), 8, 55, 67, 69
setSamFilter,GbsrGenotypeData-method
(setSamFilter), 80
showScheme, 82
showScheme(), 4, 5, 53, 56
showScheme,GbsrGenotypeData-method
(showScheme), 82
showScheme,GbsrScheme-method
(showScheme), 82

thinMarker, 83
thinMarker,GbsrGenotypeData-method
(thinMarker), 83

validMar, 84
validMar(), 20–31, 35–42, 45, 49, 50, 57, 76, 84
validMar,GbsrGenotypeData-method
(validMar), 84
validMar<- (validMar), 84
validMar<-,GbsrGenotypeData-method
(validMar), 84
validSam, 85
validSam(), 48, 58, 81
validSam,GbsrGenotypeData-method
(validSam), 85
validSam<- (validSam), 85
validSam<-,GbsrGenotypeData-method
(validSam), 85