Package 'SNPkit'

July 15, 2026

Title S4 Tools for Reading and Organizing Genetic Data

Version 0.1.2

Description Provides an integrated suite of tools for handling single nucleotide polymorphism (SNP) genotype data in large-scale genetic studies. Supports importing and merging genotype files, performing quality control on SNP markers and samples, and preparing data for downstream analyses using popular software such as 'FImpute' and 'PLINK'. Offers S4 classes and methods to efficiently encapsulate SNP data, along with utilities for generating genotype summary statistics and visualization. Additional functionalities include anticlustering approaches for batch effect control, automated script generation for external software, and streamlined workflows for large datasets commonly encountered in animal and plant breeding programs. Designed to facilitate reproducible and scalable SNP data analyses in quantitative and statistical genetics.

Depends R (>= 4.1.0)

Imports methods, ggplot2, dplyr, data.table, Rcpp, stringi, anticlust, grDevices, graphics, stats, utils, MASS, snpStats, magrittr, reshape2

LinkingTo Rcpp

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

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

License GPL-3

URL https://viniciusjunqueira.github.io/SNPkit/, https://github.com/viniciusjunqueira/SNPkit

BugReports https://github.com/viniciusjunqueira/SNPkit/issues

RoxygenNote 7.3.3

NeedsCompilation yes

Author Vinícius Junqueira [aut, cre],
Roberto Higa [aut],
Fernando Flores Cardoso [aut],
Marcos Jun Iti Yokoo [aut]

Maintainer Vinícius Junqueira <junqueiravinicius@hotmail.com>

Repository CRAN

Date/Publication 2026-07-14 23:00:02 UTC

Contents

as_snpmatrix . . . 3
check.call.rate . . . 4
check.ibs . . . 5
check.identical.samples . . . 5
check.identical.samples.by.block . . . 6
check.mendelian.inconsistencies . . . 7
check.mendelian.inconsistencies.pair . . . 7
check.sample.call.rate . . . 8
check.sample.heterozygosity . . . 9
check.snp.chromo . . . 9
check.snp.hwe . . . 10
check.snp.hwe.chi2 . . . 11
check.snp.maf . . . 12
check.snp.mgf . . . 12
check.snp.monomorf . . . 13
check.snp.no.position . . . 13
check.snp.same.position . . . 14
combineSNPData . . . 15
doPCA . . . 16
exploratory.plots . . . 16
FImputeExport-class . . . 18
FImputeRunner . . . 18
FImputeRunner-class . . . 19
genoToDF . . . 19
get.correl.fc . . . 20
get.gender . . . 21
get.hwe.chi2 . . . 21
getGeno . . . 22
ibs.pair . . . 23
importAllGenos . . . 23
importFImputeResults . . . 24
import_geno_list . . . 24
pairs2sets . . . 25
plotPCAgroups . . . 26
print.summary.SNPDataLong . . . 26
qcSamples . . . 27
qcSNPs . . . 28

qc_header 29
read.fimpute 30
runAnticlusteringPCA 31
runFImpute 32
runPCA 33
run_admixture 34
saveFImpute 35
saveFImputeRaw 36
savePlink 37
SNPFileConfig-class 38
SNPImportList-class 38
Subset 39
summary,SNPDataLong-method 39

Index **41**

`as_snpmatrix` *Convert a genotype matrix or data.frame to snpStats::SnpMatrix*

Description

This function converts a genotype matrix coded as 0/1/2/NA or AA/AB/BB to a `snpStats::SnpMatrix` object. It includes checks for coding validity, missing values, and duplicate sample or SNP IDs, and preserves row and column names from the input.

Usage

```
as_snpmatrix(
  geno,
  coding = c("012", "AAABBB"),
  missing_codes = c("NA", "-9", ".", ""),
  check_ids = TRUE
)
```

Arguments

`geno`	A samples x SNPs matrix or data.frame with genotypes coded as 0, 1, 2, or NA. Can be numeric/integer or character. `rownames` = sample IDs, `colnames` = SNP IDs.
`coding`	One of `"012"` or `"AAABBB"`. For character inputs only. `"012"` expects `"0"`, `"1"`, `"2"`, and `missing_codes`. `"AAABBB"` expects `"AA"`, `"AB"`, `"BB"`, and `missing_codes`.
`missing_codes`	Character values to treat as missing (only used when `geno` is character), e.g., `c("NA","-9",".")`.
`check_ids`	If `TRUE`, verifies that row and column names are unique (recommended).

Details

The function accepts both `matrix` and `data.frame` inputs. For `data.frame` objects, all columns are coerced to a common type using `as.matrix()`, which preserves `rownames` and `colnames`.

The returned `SnpMatrix` object stores each genotype as a single byte, which is memory-efficient compared to integer storage. However, large datasets still require substantial RAM. For very large genotype sets, consider using on-disk formats such as **SNPRelate** (GDS) or **bigsnpr**.

Value

A `snpStats::SnpMatrix` with the same `dimnames` as `geno`.

Examples

```
# Numeric 0/1/2 with NAs
set.seed(1)
geno <- matrix(sample(c(0L,1L,2L,NA), 20, replace=TRUE), nrow=5)
rownames(geno) <- paste0("ind", 1:5)
colnames(geno) <- paste0("snp", 1:4)
SM <- as_snpmatrix(geno)

# Character AA/AB/BB
geno_c <- matrix(sample(c("AA","AB","BB","."), 20, replace=TRUE,
                        prob=c(.35,.3,.3,.05)), nrow=5)
rownames(geno_c) <- rownames(geno)
colnames(geno_c) <- colnames(geno)
SMc <- as_snpmatrix(geno_c, coding="AAABBB", missing_codes=".")
```

`check.call.rate` *Check SNP call rate*

Description

Identifies SNPs with call rates below a minimum threshold.

Usage

```
check.call.rate(summary, min.call.rate)
```

Arguments

`summary`	A data frame with SNP summary statistics (must contain ‘Call.rate‘ column).
`min.call.rate`	Numeric value specifying the minimum acceptable call rate.

Value

Character vector with SNP names below threshold. Returns ‘NULL‘ if none.

Examples

```
df <- data.frame(Call.rate = c(0.85, 0.95), row.names = c("SNP1", "SNP2"))
check.call.rate(df, 0.9)
```

---

check.ibs *Check Identity-By-State (IBS) for a genotype pair*

Description

Checks IBS status for two genotypes.

Usage

```
check.ibs(gen)
```

Arguments

gen	Numeric vector of length two with genotype codes.

Value

Integer: 2 if identical non-heterozygotes, 0 if opposite homozygotes, -1 otherwise.

Examples

```
check.ibs(c(1, 1))
check.ibs(c(1, 3))
```

---

check.identical.samples *Check identical samples based on distance*

Description

Identifies sample pairs considered identical based on genotype distances.

Usage

```
check.identical.samples(genotypes, threshold = 0)
```

Arguments

genotypes	Genotype matrix (samples x SNPs) or SnpMatrix.
threshold	Numeric distance threshold. Default 0.

Value

Data frame of identical sample pairs.

Examples

```
mat <- matrix(sample(0:2, 20, TRUE), nrow = 5)
rownames(mat) <- paste0("S", 1:5)
check.identical.samples(mat, 0.5)
```

`check.identical.samples.by.block`

Check identical samples by block

Description

Identifies sample pairs that stay identical (within `threshold`) across *every* SNP block, scanning the markers in blocks of `blcsize`. Each block only re-checks the samples still in a confirmed pair, and pairs that separate in any block are dropped, so the result is the intersection of the per-block identical pairs.

Usage

```
check.identical.samples.by.block(genotypes, blcsize, threshold = 0)
```

Arguments

`genotypes`	Genotype matrix (samples x SNPs) or SnpMatrix with sample names as rownames.
`blcsize`	Block size (number of SNPs).
`threshold`	Distance threshold. Default 0.

Value

A data.frame of identical sample pairs (columns `Sample1`, `Sample2`, `Distance`); `Distance` is taken from the first block. Empty data.frame if none.

Examples

```
set.seed(1)
mat <- matrix(sample(1:3, 40, TRUE), nrow = 4)
rownames(mat) <- paste0("S", 1:4)
check.identical.samples.by.block(mat, blcsize = 5, threshold = 0)
```

check.mendelian.inconsistencies
Check Mendelian inconsistencies

Description

Identifies Mendelian inconsistencies between father-child pairs.

Usage

```
check.mendelian.inconsistencies(genotypes, father, child)
```

Arguments

genotypes	Genotype matrix.
father	Vector of father sample IDs.
child	Vector of child sample IDs.

Value

Data frame summarizing inconsistencies per pair.

Examples

```
set.seed(1)
genotypes <- matrix(sample(1:3, 30, TRUE), nrow = 3,
                    dimnames = list(c("F1", "C1", "C2"), NULL))
check.mendelian.inconsistencies(genotypes,
                                father = "F1",
                                child  = c("C1", "C2"))
```

check.mendelian.inconsistencies.pair
Check Mendelian inconsistencies for a pair

Description

Calculates number of inconsistencies and total comparable SNPs for a parent-child pair.

Usage

```
check.mendelian.inconsistencies.pair(g1, g2)
```

Arguments

g1	Genotype vector for parent.
g2	Genotype vector for child.

Value

Numeric vector: [# inconsistencies, # comparable SNPs].

Examples

```
g1 <- c(1, 1, 3, 3, 2)
g2 <- c(3, 1, 1, 3, 2)
check.mendelian.inconsistencies.pair(g1, g2)
```

---

check.sample.call.rate

Check Sample Call Rate

Description

Identifies samples with call rate below a given threshold.

Usage

```
check.sample.call.rate(sample.summary, min.call.rate)
```

Arguments

sample.summary	A data frame with a "Call.rate" column for each sample.
min.call.rate	Minimum acceptable call rate (between 0 and 1).

Value

A character vector with the names of samples to remove.

check.sample.heterozygosity *Check sample heterozygosity*

Description

Identifies samples with heterozygosity values deviating beyond a specified threshold.

Usage

```
check.sample.heterozygosity(sample.summary, max.dev)
```

Arguments

sample.summary	Data frame containing sample summary (must have 'Heterozygosity' column).
max.dev	Maximum number of standard deviations allowed from mean.

Value

Character vector with sample names considered outliers. Returns 'NULL' if none.

Examples

```
ss <- data.frame(Heterozygosity = c(0.2, 0.5, 0.7))
rownames(ss) <- c("Ind1", "Ind2", "Ind3")
check.sample.heterozygosity(ss, 1)
```

check.snp.chromo *Check SNP by chromosome*

Description

Filters SNP names belonging to specified chromosomes.

Usage

```
check.snp.chromo(snpmap, chromosomes)
```

Arguments

snpmap	Data frame with SNP map info (must contain columns 'Chromosome' and 'Name').
chromosomes	Vector of chromosome identifiers to filter.

Value

Character vector with SNP names.

Examples

```
snpmap <- data.frame(Chromosome = c(1, 1, 2), Name = c("SNP1", "SNP2", "SNP3"))
check.snp.chromo(snpmap, 1)
```

`check.snp.hwe` *Check SNP Hardy-Weinberg equilibrium deviation*

Description

Identifies SNPs deviating from HWE beyond a z-score threshold.

Usage

```
check.snp.hwe(snp.summary, max.dev)
```

Arguments

`snp.summary`	Data frame with SNP summary (must contain ‘z.HWE‘ column).
`max.dev`	Maximum z-score allowed.

Value

Character vector with SNP names deviating from HWE. Returns ‘NULL‘ if none.

Examples

```
df <- data.frame(z.HWE = c(2, 5), row.names = c("SNP1", "SNP2"))
check.snp.hwe(df, 3)
```

`check.snp.hwe.chi2` *Check SNPs for Hardy-Weinberg equilibrium deviation using chi-square p-values*

Description

This function identifies SNP markers whose Hardy-Weinberg equilibrium (HWE) chi-square p-values indicate significant deviation beyond a specified threshold. It uses the p-values computed by `get.hwe.chi2` on the input summary data frame.

Usage

```
check.snp.hwe.chi2(snp.summary, max.dev)
```

Arguments

`snp.summary`	A data frame or matrix containing summary statistics for SNP markers. The row names should correspond to SNP identifiers. It must be compatible with the function `get.hwe.chi2`.
`max.dev`	A numeric value specifying the maximum acceptable p-value threshold. SNPs with p-values below this threshold are considered as deviating from HWE.

Details

Any SNP with missing p-value (NA) is treated as not failing (returned as FALSE).

Value

A character vector of SNP identifiers (rownames) that fail the HWE test (p-value < `max.dev`). If no SNPs fail, an empty vector is returned.

See Also

`get.hwe.chi2`

Examples

```
snp.summary <- data.frame(
  Calls = c(100, 100),
  P.AA  = c(0.25, 0.7),
  P.AB  = c(0.50, 0.05),
  P.BB  = c(0.25, 0.25),
  row.names = c("SNP1", "SNP2")
)
check.snp.hwe.chi2(snp.summary, max.dev = 0.05)
```

check.snp.maf *Check SNP minor allele frequency*

Description

Identifies SNPs with minor allele frequency below a minimum threshold.

Usage

```
check.snp.maf(snp.summary, min.maf)
```

Arguments

snp.summary	Data frame with SNP summary (must contain 'MAF' column).
min.maf	Minimum MAF allowed.

Value

Character vector with SNP names below threshold. Returns 'NULL' if none.

Examples

```
df <- data.frame(MAF = c(0.01, 0.2), row.names = c("SNP1", "SNP2"))
check.snp.maf(df, 0.05)
```

check.snp.mgf *Check SNP missing genotype frequencies*

Description

Identifies SNPs with genotype frequencies below a minimum threshold.

Usage

```
check.snp.mgf(snp.summary, min.mgf)
```

Arguments

snp.summary	Data frame with columns 'P.AA', 'P.AB', 'P.BB'.
min.mgf	Minimum genotype frequency allowed.

Value

Character vector with SNP names below threshold. Returns 'NULL' if none.

Examples

```
df <- data.frame(P.AA = c(0.01, 0.5), P.AB = c(0.02, 0.4), P.BB = c(0.01, 0.1))
rownames(df) <- c("SNP1", "SNP2")
check.snp.mgf(df, 0.05)
```

---

check.snp.monomorf *Check SNP monomorphic status*

Description

Identifies SNPs considered monomorphic.

Usage

```
check.snp.monomorf(snp.summary)
```

Arguments

snp.summary	Data frame with columns 'P.AA', 'P.AB', 'P.BB'.

Value

Character vector with monomorphic SNP names. Returns 'NULL' if none.

Examples

```
df <- data.frame(P.AA = c(1, 0.5), P.AB = c(0, 0.5), P.BB = c(0, 0))
rownames(df) <- c("SNP1", "SNP2")
check.snp.monomorf(df)
```

---

check.snp.no.position *Check SNP no position*

Description

Identifies SNPs without a usable genomic position, i.e. whose position is missing ('NA'), blank, non-numeric, or zero. The 'Position' column may be numeric or character ('getGeno()' reads maps as character), so it is coerced to numeric first.

Usage

```
check.snp.no.position(snpmap)
```

Arguments

snpmap	Data frame with columns ‘Position‘ and ‘Name‘.

Value

Character vector with SNP names without position. Returns ‘NULL‘ if none.

Examples

```
df <- data.frame(Position = c(0, 100, NA), Name = c("SNP1", "SNP2", "SNP3"))
check.snp.no.position(df)  # SNP1 (zero) and SNP3 (missing)
```

---

check.snp.same.position

Check SNPs with same position

Description

Identifies SNPs that share the same position on the same chromosome.

Usage

```
check.snp.same.position(snpmap)
```

Arguments

snpmap	Data frame with columns ‘Chromosome‘, ‘Position‘, and ‘Name‘.

Value

List of SNP groups sharing positions.

Examples

```
df <- data.frame(Chromosome = c(1, 1, 2),
                 Position = c(100, 100, 200),
                 Name = c("SNP1", "SNP2", "SNP3"))
check.snp.same.position(df)
```

`combineSNPData` *Combine multiple SNPDataLong objects*

Description

This function merges a list of `SNPDataLong` objects, typically representing different SNP panels or datasets, into a single unified `SNPDataLong` object. It ensures that all genotype matrices have the same set of SNPs (filling missing SNPs with NA), and merges the marker map information while removing duplicate SNP entries.

Usage

```
combineSNPData(lista)
```

Arguments

`lista`	A list of `SNPDataLong` objects to be combined.

Value

A single `SNPDataLong` object containing the combined genotype matrix, merged map, and a concatenated path string.

Examples

```
make_obj <- function(samples, snps) {
  m <- methods::new("SnpMatrix",
                    matrix(as.raw(1:3),
                           nrow = length(samples),
                           ncol = length(snps),
                           dimnames = list(samples, snps)))
  methods::new("SNPDataLong",
               geno = m,
               map  = data.frame(Name = snps,
                                 Chromosome = 1,
                                 Position = seq_along(snps)),
               path = tempfile(),
               xref_path = "chip1")
}
obj1 <- make_obj(c("S1", "S2"), c("SNP1", "SNP2"))
obj2 <- make_obj(c("S3", "S4"), c("SNP2", "SNP3"))
combined <- combineSNPData(list(obj1, obj2))
```

doPCA *Do genome relationship matrix PCA (deprecated)*

Description

Deprecated. Performs PCA using the genome relationship matrix (GRM) on a raw `SnpMatrix`. Use `runPCA` instead, which operates on a `SNPDataLong` object, standardises SNPs, and returns scores directly comparable to `runAnticlusteringPCA`.

Usage

```
doPCA(genotypes)
```

Arguments

`genotypes`	Genotype matrix.

Value

List containing ‘pcs‘ (principal components) and ‘eigen‘ (eigenvalues).

See Also

`runPCA`

Examples

```
set.seed(1)
mat <- matrix(sample(as.raw(1:3), 200, TRUE), nrow = 10, ncol = 20)
geno <- methods::new("SnpMatrix", mat)
rownames(geno) <- paste0("S", 1:10)
colnames(geno) <- paste0("SNP", 1:20)
res <- suppressWarnings(doPCA(geno))  # doPCA is deprecated; use runPCA()
str(res)
```

exploratory.plots *Exploratory plots for SNP and sample summary*

Description

Generates exploratory plots: MAF histograms, HWE plots, heterozygosity scatter, MDS, and dendrogram.

Usage

```
exploratory.plots(
  snp.summary,
  snps.plot,
  sample.summary,
  samples.plot,
  distm,
  glabels,
  mds.plot,
  hierq.plot
)
```

Arguments

snp.summary	Data frame with SNP summary.
snps.plot	Filename for SNP histogram plot.
sample.summary	Data frame with sample summary.
samples.plot	Filename for heterozygosity plot.
distm	Distance matrix for samples.
glabels	Sample labels for plots.
mds.plot	Filename for MDS plot.
hierq.plot	Filename for hierarchical cluster plot.

Value

NULL (plots are saved as JPEG files)

Examples

```
tmp <- tempfile(fileext = ".jpg")
snp.summary <- data.frame(
  MAF   = runif(20),
  z.HWE = rnorm(20),
  Calls = rep(100, 20),
  P.AA  = runif(20, 0, 0.5),
  P.AB  = runif(20, 0, 0.5),
  P.BB  = runif(20, 0, 0.5)
)
sample.summary <- data.frame(
  Call.rate      = runif(5, 0.9, 1),
  Heterozygosity = runif(5, 0.2, 0.4),
  row.names = paste0("S", 1:5)
)
distm <- stats::dist(matrix(rnorm(25), nrow = 5))
exploratory.plots(snp.summary,
                  snps.plot      = tempfile(fileext = ".jpg"),
                  sample.summary = sample.summary,
                  samples.plot   = tempfile(fileext = ".jpg"),
```

```
distm          = distm,
glabels        = paste0("S", 1:5),
mds.plot       = tempfile(fileext = ".jpg"),
hierq.plot     = tempfile(fileext = ".jpg"))
```

`FImputeExport-class` *FImputeExport Class*

Description

A class to handle export preparation for FImpute.

Slots

`geno` A SnpMatrix or NULL containing genotype data.

`map` A data.frame containing marker information.

`path` Output file path.

`name` Project or file name.

`FImputeRunner` *Build FImputeRunner object*

Description

A convenience function to construct a ‘FImputeRunner‘ object from a ‘SNPDataLong‘ object.

Usage

```
FImputeRunner(object, path, exec_path = "FImpute3", name = "data")
```

Arguments

`object`	An object of class ‘SNPDataLong‘, from which ‘geno‘ and ‘map‘ slots will be extracted.
`path`	A character string indicating the directory to save FImpute files.
`exec_path`	Path to the FImpute executable (default = "FImpute3").
`name`	Name for the dataset (used internally, default = "data").

Value

An object of class ‘FImputeRunner‘.

FImputeRunner-class *FImputeRunner Class*

Description

A class to manage FImpute execution and results.

Slots

`export` An FImputeExport object.

`par_file` Path to parameter file.

`exec_path` Path to FImpute executable.

`results` A data.frame containing results or summary information.

genoToDF *Convert geno slot from SNPDataLong to a data.frame*

Description

Converts the genotype matrix (geno slot) of a SNPDataLong object to a data.frame, with optional centering and scaling per SNP (column).

Usage

```
genoToDF(object, center = FALSE, scale = FALSE)
```

Arguments

`object`	An object of class SNPDataLong.
`center`	Logical or numeric. If TRUE (default FALSE), center columns to mean zero.
`scale`	Logical or numeric. If TRUE (default FALSE), scale columns to standard deviation one.

Value

A data.frame with individuals as rows and SNPs as columns (numeric 0/1/2, or centered/scaled values).

Examples

```
set.seed(1)
raw_mat <- matrix(as.raw(sample(1:3, 100, TRUE)), nrow = 10, ncol = 10)
rownames(raw_mat) <- paste0("S", 1:10)
colnames(raw_mat) <- paste0("SNP", 1:10)
geno <- methods::new("SnpMatrix", raw_mat)
obj <- methods::new("SNPDataLong",
                    geno = geno,
                    map  = data.frame(Name = colnames(geno),
                                      Chromosome = 1,
                                      Position = 1:10),
                    path = tempfile(),
                    xref_path = "chip1")
df <- genoToDF(obj, center = TRUE, scale = TRUE)
head(df[, 1:5])
```

---

get.correl.fc *Get correlation (fc method)*

Description

Calculates genotype correlation using a fast check (fc) method.

Usage

```
get.correl.fc(g1, g2)
```

Arguments

g1	Genotype vector.
g2	Genotype vector.

Value

Numeric value of correlation.

Examples

```
g1 <- sample(0:2, 10, TRUE)
g2 <- sample(0:2, 10, TRUE)
get.correl.fc(g1, g2)
```

get.gender *Get gender based on heterozygosity*

Description

Infers gender using heterozygosity thresholds.

Usage

```
get.gender(sample.summary, threshM, threshF)
```

Arguments

sample.summary	Data frame with ‘Heterozygosity‘ column.
threshM	Numeric threshold for males.
threshF	Numeric threshold for females.

Value

Data frame with columns ‘heterozygosity‘ and ‘sex‘.

Examples

```
df <- data.frame(Heterozygosity = c(0.1, 0.3, 0.6))
rownames(df) <- c("A", "B", "C")
get.gender(df, 0.2, 0.5)
```

get.hwe.chi2 *Get HWE chi-square p-values*

Description

Calculates Hardy-Weinberg equilibrium chi-square p-values for SNPs.

Usage

```
get.hwe.chi2(snp.summary)
```

Arguments

snp.summary	Data frame with columns ‘Calls‘, ‘P.AA‘, ‘P.AB‘, ‘P.BB‘.

Value

Numeric vector with p-values.

Examples

```
df <- data.frame(Calls = c(100, 100), P.AA = c(0.6, 0.4), P.AB = c(0.3, 0.4), P.BB = c(0.1, 0.2))
get.hwe.chi2(df)
```

---

getGeno *Flexible and efficient genotype file reading using fread*

Description

Imports SNP genotype data from Illumina FinalReport files using `data.table::fread` and builds the `SnpMatrix` directly from the long-format calls. This is reliable even for very large files (millions of lines, hundreds of samples), where `snpStats::read.snps.long` may fail to read all samples. Empty or unreadable confidence values are treated as no calls. The original file on disk is never modified.

Usage

```
getGeno(...)

## S4 method for signature 'ANY'
getGeno(
  path,
  fields = list(sample = 2, snp = 1, allele1 = 7, allele2 = 8, confidence = 9),
  codes = c("A", "B"),
  threshold = 0.15,
  sep = "\t",
  skip = 0,
  verbose = TRUE,
  every = NULL
)
```

Arguments

`...`	Additional optional arguments.
`path`	Path to the directory containing `FinalReport.txt`
`fields`	List specifying column indices (sample, snp, allele1, allele2, confidence)
`codes`	Allele codes (e.g., `c("A", "B")`); a genotype is coded as the count of `codes[2]` alleles (homozygous `codes[1]`, heterozygous, homozygous `codes[2]`).
`threshold`	Confidence threshold; calls below it are set to missing
`sep`	Field separator
`skip`	Lines to skip
`verbose`	Logical; show progress
`every`	Deprecated; kept for backward compatibility and ignored.

Value

An SNPDataLong object

---

ibs.pair *IBS pair statistics*

Description

Calculates IBS mean and standard deviation between two samples.

Usage

```
ibs.pair(g1, g2)
```

Arguments

g1	Genotype vector for first sample.
g2	Genotype vector for second sample.

Value

Numeric vector: [mean IBS, standard deviation].

Examples

```
g1 <- sample(0:2, 10, TRUE)
g2 <- sample(0:2, 10, TRUE)
ibs.pair(g1, g2)
```

---

importAllGenos *Import and combine multiple genotype configurations*

Description

Imports genotype data from multiple configurations defined in an SNPImportList object and combines them into a unified SNPDataLong object.

Usage

```
importAllGenos(object)

## S4 method for signature 'SNPImportList'
importAllGenos(object)
```

Arguments

object	An SNPImportList object.

Value

A combined SNPDataLong object.

---

importFImputeResults *Import imputed FImpute results from disk*

Description

Reads existing imputed results from a given path and returns an object of class SNPDataLong.

Usage

```
importFImputeResults(path, method = "R")
```

Arguments

path	Character. Path to the folder containing 'output_fimpute' (e.g., "fimpute_run_nelore").
method	Character. "R" (default) or "Rcpp". Passed to read.fimpute().

Value

An object of class SNPDataLong containing the imputed genotypes and SNP map.

---

import_geno_list *Import multiple genotype datasets from a list of configurations*

Description

Reads and imports multiple genotype datasets specified in a list of configurations. Each configuration must include the path to the genotype data and information on field mapping. Optionally, you can also specify codes, quality threshold, separator, lines to skip, and a subset of IDs to retain. The function automatically fills the 'xref_path' slot per individual and combines maps into a single data.frame, adding a 'SourcePath' column indicating their origin and removing duplicated SNP rows (by Name). Prints progress messages indicating the current path being loaded (with counter).

Usage

```
import_geno_list(config_list)
```

Arguments

config_list	A list of configuration lists. Each element should contain: - ‘path‘ (character): Path to the genotype file or folder. - ‘fields‘ (list): Named list defining the columns (e.g., SNP ID, sample ID, alleles, confidence). - ‘codes‘ (character vector, optional): Allele codes (default is c("A", "B")). - ‘threshold‘ (numeric, optional): Maximum allowed missingness or confidence threshold (default 0.15). - ‘sep‘ (character, optional): Field separator in the input file (default "tab-delimited"). - ‘skip‘ (integer, optional): Number of lines to skip at the beginning of the file (default 0). - ‘verbose‘ (logical, optional): Whether to print detailed messages (default TRUE). - ‘subset‘ (character vector, optional): Vector of sample IDs to retain after import.

Value

An object of class ‘SNPDataLong‘ containing: - Combined genotype matrix (‘geno‘). - Combined map (‘map‘) as a single data.frame with ‘SourcePath‘ column and without duplicated rows. - Combined ‘xref_path‘ vector (one entry per individual). - ‘path‘ slot as a semicolon-separated string of all input dataset paths.

---

pairs2sets *Convert pairs to sets*

Description

Groups sample pairs into sets of related samples.

Usage

```
pairs2sets(pairs)
```

Arguments

pairs	Matrix or list of sample pairs.

Value

List of sets of samples.

Examples

```
pairs <- matrix(c("A", "B", "B", "C", "D", "E"), ncol = 2, byrow = TRUE)
pairs2sets(pairs)
```

plotPCAgroups *Plot PCA groups from anticlustering result*

Description

Plot PCA groups from anticlustering result

Usage

```
plotPCAgroups(pca_res, groups, pcs = c(1, 2), filename = NULL)
```

Arguments

pca_res	A prcomp object.
groups	A factor or vector of group assignments.
pcs	Vector of length 2 indicating which PCs to plot (default: c(1, 2)).
filename	Optional. If provided, saves plot to this file (e.g., "antic.png").

Value

A ggplot object (also prints to screen).

Examples

```
set.seed(1)
pca_res <- stats::prcomp(matrix(rnorm(200), nrow = 20))
groups <- sample(1:2, 20, replace = TRUE)
plotPCAgroups(pca_res, groups)
```

print.summary.SNPDataLong *Print method for SNPDataLong summary*

Description

Displays the contents of a summary.SNPDataLong object on the console.

Usage

```
## S3 method for class 'summary.SNPDataLong'
print(x, ...)
```

Arguments

x	An object of class summary.SNPDataLong.
...	Further arguments (currently unused).

Value

The input x, returned invisibly.

---

qcSamples *Quality control on samples*

Description

Applies quality control (QC) procedures to samples in a ‘SNPDataLong‘ object, based on heterozygosity and call rate thresholds.

Usage

```
qcSamples(x, ...)

## S4 method for signature 'SNPDataLong'
qcSamples(
  x,
  heterozygosity = NULL,
  smp_cr = NULL,
  action = c("report", "filter", "both")
)
```

Arguments

x	An object of class ‘SNPDataLong‘.
...	Additional optional arguments.
heterozygosity	A numeric threshold or range for heterozygosity. Samples outside this threshold are removed.
smp_cr	Minimum acceptable sample call rate (between 0 and 1). Samples below this value are removed.
action	Character string indicating the action to perform. One of: - ‘"report"‘: only returns a list of samples to remove and those kept; - ‘"filter"‘: returns a filtered object without reporting; - ‘"both"‘: performs filtering and returns the filtered object.

Value

Depending on the ‘action‘ argument: - ‘"report"‘: returns a list with removed and kept samples; - ‘"filter"‘: returns a new ‘SNPDataLong‘ object with filtered genotypes; - ‘"both"‘: returns a list with: - ‘filtered‘: the filtered ‘SNPDataLong‘ object; - ‘report‘: a list of removed and kept samples.

qcSNPs *Quality Control for SNPDataLong with optional criteria*

Description

Applies flexible quality control filters on an object of class SNPDataLong. Supports call rate filtering, minor allele frequency (MAF), Hardy-Weinberg equilibrium (HWE), removal of monomorphic SNPs, exclusion of specific chromosomes, optionally removing SNPs without positions, and optionally removing SNPs at the same genomic position (keeping the one with highest MAF).

Usage

```
qcSNPs(x, ...)

## S4 method for signature 'SNPDataLong'
qcSNPs(
  x,
  min_snp_cr = NULL,
  min_maf = NULL,
  hwe = NULL,
  snp_position = NULL,
  no_position = NULL,
  snp_mono = FALSE,
  remove_chr = NULL,
  action = c("report", "filter", "both")
)
```

Arguments

x	An object of class SNPDataLong.
...	Additional optional arguments.
min_snp_cr	Minimum acceptable call rate for SNPs (e.g., 0.95). SNPs below this threshold are removed. For per-individual missingness use qcSamples(smp_cr = ...).
min_maf	Minimum minor allele frequency allowed for SNPs (e.g., 0.05). SNPs with lower MAF are removed.
hwe	p-value threshold for Hardy-Weinberg equilibrium test (e.g., 1e-6). SNPs violating this are removed.
snp_position	Logical. If TRUE, removes SNPs mapped to the same position, retaining only the one with highest MAF.
no_position	Logical. If TRUE, removes SNPs without defined genomic positions.
snp_mono	Logical. If TRUE, removes monomorphic SNPs (with no variation).
remove_chr	Character vector of chromosomes to exclude (e.g., c("X", "Y")).
action	One of "report" (returns a list of removed SNPs), "filter" (returns filtered SNPDataLong), or "both" (returns both).

Value

Depending on the action argument: - "report": list of SNPs removed by each filter and SNPs retained. - "filter": filtered SNPDataLong object. - "both": list containing the filtered object and detailed report.

Examples

```
set.seed(123)
raw_mat <- matrix(as.raw(sample(1:3, 100, TRUE)), nrow = 10, ncol = 10)
colnames(raw_mat) <- paste0("snp", 1:10)
rownames(raw_mat) <- paste0("ind", 1:10)
geno <- methods::new("SnpMatrix", raw_mat)
map <- data.frame(Name = colnames(geno), Chromosome = 1, Position = 1:10)
x <- methods::new("SNPDataLong",
                  geno = geno,
                  map  = map,
                  path = tempfile(),
                  xref_path = "chip1")

qcSNPs(x,
       min_snp_cr = 0.8,
       min_maf = 0.05,
       snp_mono = TRUE,
       no_position = TRUE,
       snp_position = TRUE,
       action = "filter")
```

---

qc_header *Formatted header message*

Description

Prints a formatted message with a border for section titles in the console.

Usage

```
qc_header(title)
```

Arguments

title	Character string to be printed inside the header box.

Value

No return value. Used for side effects (message).

Examples

```
qc_header("Quality Control on Samples")
```

`read.fimpute` *Read imputed genotypes from FImpute output and return SNPDataLong object*

Description

Reads imputed genotypes and SNP information from FImpute output, builds a SnpMatrix and a corresponding map, and returns an SNPDataLong object.

Usage

```
read.fimpute(file, method = c("R", "Rcpp"))
```

Arguments

`file`	Character. Path to the FImpute output directory (usually "output_fimpute").
`method`	Character. "R" (default) for vectorized R implementation, or "Rcpp" for compiled C++ implementation.

Value

An object of class `SNPDataLong` with three slots: `geno` (a `SnpMatrix` with individuals as rows and SNPs as columns), `map` (a `data.frame` with columns `Name`, `Chromosome`, and `Position`), and `path` (the input directory).

Examples

```
## Not run:
# Requires a directory containing FImpute output files
# (genotypes_imp.txt and snp_info.txt).
snp_long <- read.fimpute("output_fimpute", method = "R")

## End(Not run)
```

`runAnticlusteringPCA` *Run PCA and anticlustering on SNPDataLong*

Description

Runs PCA on a `SNPDataLong` object (via `runPCA`) and then performs anticlustering on the selected principal components.

Usage

```
runAnticlusteringPCA(
  object,
  K = 2,
  n_pcs = 20,
  center = TRUE,
  scale = TRUE,
  anticlust_method = c("exchange", "fast")
)
```

Arguments

`object`	An object of class `SNPDataLong`.
`K`	Number of groups for anticlustering, or a vector of group sizes (as in **anticlust**).
`n_pcs`	Number of top principal components to use. If `< 1`, it is interpreted as the proportion of variance to be explained (e.g. `0.8`). The fast matrix-free PCA path is used when a fixed number (`>= 1`) is requested and **RSpectra** is installed.
`center`	Logical or numeric, passed to `runPCA`. Default `TRUE`.
`scale`	Logical or numeric, passed to `runPCA`. Default `TRUE`.
`anticlust_method`	Which **anticlust** optimiser to use. `"exchange"` (default) calls `anticlust::anticlustering`; `"fast"` calls `anticlust::fast_anticlustering`, which scales to large numbers of individuals and may return different assignments.

Value

A list with components:

groups Integer vector with anticlustering group assignments.

pca The PCA result object (a `prcomp`-like list), as returned by `runPCA`.

pcs Numeric matrix of the PCs used for anticlustering.

Examples

```
res <- runAnticlusteringPCA(nelore_imputed, K = 2, n_pcs = 0.8)
table(res$groups)
```

runFImpute *Run FImpute from a FImputeRunner object*

Description

This function runs the external FImpute software using a ‘FImputeRunner‘ object, ensuring that all required input files are present and the results are imported.

Usage

```
runFImpute(object, verbose = TRUE)

## S4 method for signature 'FImputeRunner'
runFImpute(object, verbose = TRUE)
```

Arguments

object	An object of class ‘FImputeRunner‘.
verbose	Logical. If TRUE (default), FImpute output will be printed to the console.

Value

An updated ‘FImputeRunner‘ object with the ‘results‘ slot populated (SNPDataLong).

Examples

```
## Not run:
# Requires the external FImpute3 binary in PATH.
path_fimpute <- file.path(tempdir(), "fimpute_run_example")
param_file <- file.path(path_fimpute, "fimpute.par")

export_obj <- methods::new("FImputeExport",
                           geno = geno_obj@geno,
                           map  = geno_obj@map,
                           path = path_fimpute)

runner <- methods::new("FImputeRunner",
                       export    = export_obj,
                       par_file  = param_file,
                       exec_path = "FImpute3")

res <- runFImpute(runner, verbose = TRUE)

## End(Not run)
```

runPCA *Run PCA on a SNPDataLong object*

Description

Computes principal components of the (optionally centered/scaled) genotype matrix, without any clustering. For wide data (more SNPs than individuals) the PCA is obtained from the n x n Gram matrix, so the large rotation matrix is never formed; when a fixed number of PCs is requested and **RSpectra** is installed, only the top PCs are computed with a matrix-free solver. This is the same PCA engine used by `runAnticlusteringPCA`, so the scores are directly comparable.

Usage

```
runPCA(object, n_pcs = NULL, center = TRUE, scale = TRUE)
```

Arguments

`object`	An object of class `SNPDataLong`.
`n_pcs`	Number of principal components to return. `NULL` (default) returns all PCs; a value `>= 1` returns that many and enables the fast **RSpectra** path; a value `< 1` is the proportion of variance to reach (e.g. `0.98`).
`center`	Logical or numeric. Passed to `scale`. If `TRUE`, center columns; if numeric, a vector of column means. Default: `TRUE`.
`scale`	Logical or numeric. Passed to `scale`. If `TRUE`, scale to unit variance; if numeric, a vector of column sds. Default: `TRUE`.

Value

A list with components:

pca A `prcomp`-like object: `sdev`, `x` (scores) and `totvar` (total column variance). `rotation` is `NULL` for the wide-data paths.

pcs Numeric matrix with the selected top principal components.

Examples

```
pr <- runPCA(nelore_imputed, n_pcs = 10)
head(pr$pcs[, 1:2])
```

`run_admixture` *Run ADMIXTURE analysis*

Description

This function runs the ADMIXTURE program on a set of PLINK files (.bed/.bim/.fam) located in a specified directory, using a given file prefix. It supports both unsupervised and supervised analyses, optional cross-validation, and custom output file prefixes to avoid overwriting results.

Usage

```
run_admixture(
  path,
  prefix,
  admixture_path = "admixture",
  K,
  supervised = FALSE,
  pop_assignments = NULL,
  extra_args = NULL,
  out_prefix = NULL,
  cv = NULL
)
```

Arguments

`path`	Character. Path to the folder containing PLINK files.
`prefix`	Character. File prefix (without extension). The function will look for '<prefix>.bed', '<prefix>.bim', and '<prefix>.fam' in 'path'.
`admixture_path`	Character. Path to the ADMIXTURE executable, or "admixture" if in system PATH. Default is "admixture".
`K`	Integer. Number of ancestral populations to estimate.
`supervised`	Logical. If TRUE, runs ADMIXTURE in supervised mode (requires `pop_assignments`). Default is FALSE.
`pop_assignments`	Character vector. Population assignments for each individual (length equal to number of individuals in '.fam'). Use `NA` or "-" for missing. Required if `supervised = TRUE`.
`extra_args`	Character vector. Additional arguments to pass to ADMIXTURE (e.g., other flags). Default is NULL.
`out_prefix`	Character. Optional prefix for renaming output files (.Q, .P, .log) after the run completes. Default is NULL.
`cv`	Integer. Number of folds for cross-validation (e.g., 5 or 10). If provided, adds `--cv=cv`. Default is NULL.

Details

When supervised = TRUE, a '.pop' file is automatically created in the specified directory. Each line in this file corresponds to one individual, containing the population name or "-" for missing assignments.

If out_prefix is provided, the function renames the standard ADMIXTURE output files (e.g., '<prefix>.3.Q') to use this prefix (e.g., 'myrun.Q').

The function only works on Linux or macOS systems.

Value

No value returned. Runs ADMIXTURE as a side effect. Generates output files in the specified directory. Messages indicate progress and output file names.

Examples

```
## Not run:
# Requires the external ADMIXTURE binary and PLINK files prepared beforehand.
work_dir <- file.path(tempdir(), "admixture_demo")
run_admixture(
  path = work_dir,
  prefix = "plink_data",
  admixture_path = "admixture",
  K = 3,
  out_prefix = "run1_k3"
)

pop_vec <- c("A", "A", "B", "B", "-", "-", "A", "B", "A", "-")
run_admixture(
  path = work_dir,
  prefix = "plink_data",
  admixture_path = "admixture",
  K = 3,
  supervised = TRUE,
  pop_assignments = pop_vec,
  cv = 10,
  out_prefix = "supervised_k3_cv10"
)

## End(Not run)
```

---

saveFImpute *Save genotype and map files in FImpute format*

Description

S4 method to export genotype (.gen), map (.map), and parameter (fimpute.par) files compatible with [FImpute](https://www.aps.uoguelph.ca/~msargol/fimpute/).

Usage

```
saveFImpute(object, ...)

## S4 method for signature 'FImputeExport'
saveFImpute(object)

## S4 method for signature 'SNPDataLong'
saveFImpute(object, path)
```

Arguments

object	An object of class 'FImputeExport' or 'SNPDataLong'.
...	Additional arguments passed to methods.
path	Output directory. Must be supplied by the caller (e.g. a path inside tempdir() for examples).

Value

No return value, called for side effects. The function writes the files data.gen, data.map, and fimpute.par to the directory path and returns NULL invisibly.

---

saveFImputeRaw *Export genotypes and map using basic arguments*

Description

Convenience function to export FImpute files directly from a 'SnpMatrix' and map 'data.frame'.

Usage

```
saveFImputeRaw(geno, map, path, xref = NULL)
```

Arguments

geno	A 'SnpMatrix' object.
map	A data.frame with columns 'Name', 'Chromosome', 'Position', and 'SourcePath'.
path	Output directory.
xref	Optional vector of identifiers per individual (used to assign numeric chip IDs).

Value

No return value, called for side effects. The function writes three files (data.gen, data.map, and fimpute.par) to the directory specified by path and returns NULL invisibly.

savePlink *Save SNPDataLong object to PLINK format*

Description

Saves genotype and map data from an SNPDataLong object in PLINK format (.ped/.map and optionally binary files).

Usage

```
savePlink(
  object,
  path,
  name = "plink_data",
  run_plink = TRUE,
  chunk_size = 1000,
  extra_args = NULL
)
```

Arguments

object	An object of class SNPDataLong.
path	Character. Directory where files will be saved. Must be supplied by the caller (e.g. a folder inside `tempdir()` for examples).
name	Character. Base name for PLINK output files.
run_plink	Logical. If TRUE (default), runs PLINK1 to convert to binary files. If FALSE, only .ped and .map files are saved.
chunk_size	Integer. Number of individuals per chunk for writing .ped file (default: 1000).
extra_args	Character vector. Extra arguments appended verbatim to the PLINK command line when `run_plink = TRUE`. Useful for non-human data; e.g. `"--chr-set 29"` makes PLINK treat autosomes 1-29 correctly for cattle instead of reading 23-26 as human X/Y/XY/MT. Default is NULL.

Value

No return value, called for side effects. Files (`.ped`/`.map`, and `.bed`/`.bim`/`.fam` when `run_plink = TRUE`) are written under `path`.

Examples

```
set.seed(1)
raw_mat <- matrix(as.raw(sample(1:3, 100, TRUE)), nrow = 10, ncol = 10)
rownames(raw_mat) <- paste0("S", 1:10)
colnames(raw_mat) <- paste0("SNP", 1:10)
geno <- methods::new("SnpMatrix", raw_mat)
obj <- methods::new("SNPDataLong",
```

```
                    geno = geno,
                    map  = data.frame(Name = colnames(geno),
                                      Chromosome = 1,
                                      Position = 1:10),
                    path = tempfile(),
                    xref_path = "chip1")
savePlink(obj, path = tempdir(), name = "demo",
          run_plink = FALSE, chunk_size = 5)
```

SNPFileConfig-class *SNPFileConfig Class*

Description

A class for configuring SNP file import options.

Slots

`path` Path to the SNP file.

`fields` A list specifying column mappings or field configurations.

`codes` Character vector for genotype or allele codes.

`threshold` Numeric value for filtering or quality control.

`sep` Character specifying the field separator.

`skip` Number of lines to skip at the top of the file.

SNPImportList-class *SNPImportList Class*

Description

A class for managing a list of SNP file import configurations.

Slots

`configs` A list of SNPFileConfig objects.

Subset *Subset an SNPDataLong object*

Description

Subsets an SNPDataLong object by rows (individuals) or columns (SNPs). You can specify which individuals or SNP markers to keep or remove.

Usage

```
Subset(object, index, margin = 1, keep = TRUE)

## S4 method for signature 'SNPDataLong'
Subset(object, index, margin = 1, keep = TRUE)
```

Arguments

object	A SNPDataLong object.
index	Character vector with row (individual) or column (SNP) names to filter.
margin	Integer: 1 = rows (individuals), 2 = columns (SNPs).
keep	Logical; if TRUE, keeps the specified names; if FALSE, removes them.

Value

A new SNPDataLong object, subsetted accordingly.

summary,SNPDataLong-method *Summary for SNPDataLong objects*

Description

Provides a detailed summary of an SNPDataLong object, including sample and SNP counts, proportion of missing data, and SNP distribution by chromosome if mapping information is available.

Usage

```
## S4 method for signature 'SNPDataLong'
summary(object, ...)
```

Arguments

object	An object of class SNPDataLong.
...	Further arguments passed to methods.

Value

An object of class `summary.SNPDataLong`, which is a list with the following elements:

n_individuals Integer. Number of individuals (rows of `geno`).

n_snps Integer. Number of SNPs (columns of `geno`).

n_missing Integer. Total number of missing genotype calls.

prop_missing Numeric. Proportion of missing genotype calls.

by_chromosome Either a table of SNP counts per chromosome (when the map provides `Name` and `Chromosome`) or `NULL`.

missing_by_chromosome Either a table of SNPs with at least one missing call per chromosome, or `NULL`.

The object also has a dedicated `print` method that displays the summary on the console.

Index

as_snpmatrix, 3

check.call.rate, 4
check.ibs, 5
check.identical.samples, 5
check.identical.samples.by.block, 6
check.mendelian.inconsistencies, 7
check.mendelian.inconsistencies.pair, 7
check.sample.call.rate, 8
check.sample.heterozygosity, 9
check.snp.chromo, 9
check.snp.hwe, 10
check.snp.hwe.chi2, 11
check.snp.maf, 12
check.snp.mgf, 12
check.snp.monomorf, 13
check.snp.no.position, 13
check.snp.same.position, 14
combineSNPData, 15

doPCA, 16

exploratory.plots, 16

FImputeExport-class, 18
FImputeRunner, 18
FImputeRunner-class, 19

genoToDF, 19
get.correl.fc, 20
get.gender, 21
get.hwe.chi2, *11*, 21
getGeno, 22
getGeno,ANY-method *(*getGeno*)*, 22

ibs.pair, 23
import_geno_list, 24
importAllGenos, 23
importAllGenos,SNPImportList-method *(*importAllGenos*)*, 23
importFImputeResults, 24

pairs2sets, 25
plotPCAgroups, 26
print.summary.SNPDataLong, 26

qc_header, 29
qcSamples, 27
qcSamples,SNPDataLong-method *(*qcSamples*)*, 27
qcSNPs, 28
qcSNPs,SNPDataLong-method *(*qcSNPs*)*, 28

read.fimpute, 30
run_admixture, 34
runAnticlusteringPCA, *16*, 31, *33*
runFImpute, 32
runFImpute,FImputeRunner-method *(*runFImpute*)*, 32
runPCA, *16*, *31*, 33

saveFImpute, 35
saveFImpute,FImputeExport-method *(*saveFImpute*)*, 35
saveFImpute,SNPDataLong-method *(*saveFImpute*)*, 35
saveFImputeRaw, 36
savePlink, 37
scale, *33*
SNPFileConfig-class, 38
SNPImportList-class, 38
Subset, 39
Subset,SNPDataLong-method *(*Subset*)*, 39
summary,SNPDataLong-method, 39