Package: BIGr (via r-universe)

May 18, 2026

Title Breeding Insight Genomics Functions for Polyploid and Diploid Species

Version 0.7.2

Maintainer Alexander M. Sandercock <sandercock.alex@gmail.com>

Description Functions developed within Breeding Insight to analyze diploid and polyploid breeding and genetic data. 'BIGr' provides the ability to filter variant call format (VCF) files, extract single nucleotide polymorphisms (SNPs) from diversity arrays technology missing allele discovery count (DArT MADC) files, and manipulate genotype data for both diploid and polyploid species. It also serves as the core dependency for the 'BIGapp' 'Shiny' app, which provides a user-friendly interface for performing routine genotype analysis tasks such as dosage calling, filtering, principal component analysis (PCA), genome-wide association studies (GWAS), and genomic prediction. For more details about the included 'breedTools' functions, see Funkhouser et al. (2017) <doi:10.2527/tas2016.0003>, and the 'updog' output format, see Gerard et al. (2018) <doi:10.1534/genetics.118.301468>.

License Apache License (>= 2)

URL https://github.com/Breeding-Insight/BIGr

BugReports https://github.com/Breeding-Insight/BIGr/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.4.0)

Imports parallel, dplyr, Rdpack (>= 0.7), readr (>= 2.1.5), reshape2 (>= 1.4.4), rlang, tidyr (>= 1.3.1), vcfR (>= 1.15.0), Rsamtools, Biostrings, pwalign, janitor, quadprog, tibble, stringr, data.table

Suggests covr, ggplot2, spelling, rmdformats, knitr (>= 1.10), rmarkdown, polyRAD, testthat (>= 3.0.0)

RdMacros Rdpack

Config/pak/sysreqs libbz2-dev libicu-dev liblzma-dev libx11-dev xz-utils zlib1g-dev

Repository https://breeding-insight.r-universe.dev

Date/Publication 2026-05-18 12:03:32 UTC

RemoteUrl https://github.com/breeding-insight/bigr

RemoteRef HEAD

RemoteSha 12798c2f6dc9da58e41a783f7f01fba1d85989ad

Contents

`allele_freq_poly` *Compute Allele Frequencies for Populations*

Description

Computes allele frequencies for specified populations given SNP array data

Usage

```
allele_freq_poly(geno, populations, ploidy = 2)
```

Arguments

geno	matrix of genotypes coded as the dosage of allele B {0, 1, 2, ..., ploidy} with individuals in rows (named) and SNPs in columns (named)
populations	list of named populations. Each population has a vector of IDs that belong to the population. Allele frequencies will be derived from all animals
ploidy	integer indicating the ploidy level (default is 2 for diploid)

Value

data.frame consisting of allele_frequencies for populations (columns) for each SNP (rows)

References

Funkhouser SA, Bates RO, Ernst CW, Newcom D, Steibel JP. Estimation of genome-wide and locus-specific breed composition in pigs. Transl Anim Sci. 2017 Feb 1;1(1):36-44.

Examples

```
# Example inputs
geno_matrix <- matrix(
c(4, 1, 4, 0, # S1
  2, 2, 1, 3, # S2
  0, 4, 0, 4, # S3
  3, 3, 2, 2, # S4
  1, 4, 2, 3),# S5
nrow = 4, ncol = 5, byrow = FALSE, # individuals=rows, SNPs=cols
dimnames = list(paste0("Ind", 1:4), paste0("S", 1:5))
)

pop_list <- list(
PopA = c("Ind1", "Ind2"),
PopB = c("Ind3", "Ind4")
)

allele_freqs <- allele_freq_poly(geno = geno_matrix, populations = pop_list, ploidy = 4)
print(allele_freqs)
```

`calculate_Het` *Calculate Observed Heterozygosity from a Genotype Matrix*

Description

This function calculates the observed heterozygosity from a genotype matrix. It assumes that the samples are the columns, and the genomic markers are in rows. Missing data should be set as NA, which will then be ignored for the calculations. All samples must have the same ploidy.

Usage

```
calculate_Het(geno, ploidy)
```

Arguments

`geno`	Genotype matrix or data.frame
`ploidy`	The ploidy of the species being analyzed

Value

A dataframe of observed heterozygosity values for each sample

Examples

```
# example input for a diploid
geno <- data.frame(
            Sample1 = c(0, 1, 2, NA, 0),
            Sample2 = c(1, 1, 2, 0, NA),
            Sample3 = c(0, 1, 1, 0, 2),
            Sample4 = c(0, 0, 1, 1, NA)
           )
row.names(geno) <- c("Marker1", "Marker2", "Marker3", "Marker4", "Marker5")

ploidy <- 2

# calculate observed heterozygosity
result <- calculate_Het(geno, ploidy)

print(result)
```

`calculate_MAF` *Calculate Minor Allele Frequency from a Genotype Matrix*

Description

This function calculates the allele frequency and minor allele frequency from a genotype matrix. It assumes that the Samples are the columns, and the genomic markers are in rows. Missing data should be set as NA, which will then be ignored for the calculations. All samples must have the same ploidy.

Usage

```
calculate_MAF(df, ploidy)
```

Arguments

`df`	Genotype matrix or data.frame
`ploidy`	The ploidy of the species being analyzed

Value

A dataframe of AF and MAF values for each marker

Examples

```
# example input for a diploid
geno <- data.frame(
            Sample1 = c(0, 1, 2, NA, 0),
            Sample2 = c(1, 1, 2, 0, NA),
            Sample3 = c(0, 1, 1, 0, 2),
            Sample4 = c(0, 0, 1, 1, NA)
           )
row.names(geno) <- c("Marker1", "Marker2", "Marker3", "Marker4", "Marker5")

ploidy <- 2

# calculate allele frequency
result <- calculate_MAF(geno, ploidy)

print(result)
```

`check_homozygous_trios` *Check Homozygous Loci in Trios*

Description

This function analyzes homozygous loci segregation in trios (parents and progeny) using genotype data from a VCF file. It calculates the percentage of homozygous loci in the progeny that match the expected segregation patterns based on the tested parents.

Usage

```
check_homozygous_trios(
  path.vcf,
  ploidy = 4,
  parents_candidates = NULL,
  progeny_candidates = NULL,
  verbose = TRUE
)
```

Arguments

`path.vcf`	A string specifying the path to the VCF file containing genotype data.
`ploidy`	An integer specifying the ploidy level of the samples. Default is 4.
`parents_candidates`	A character vector of parent sample names to be tested. Must be provided.
`progeny_candidates`	A character vector of progeny sample names to be tested. Must be provided.
`verbose`	A logical value indicating whether to print the number of combinations tested. Default is TRUE.

Details

This function is designed to validate the segregation of homozygous loci in trios, ensuring that the progeny genotypes align with the expected patterns based on the parental genotypes. It requires both parent and progeny candidates to be specified. The function validates the ploidy level and ensures that all specified samples are present in the VCF file. The results include detailed statistics for each combination of parents and progeny. Reciprocal comparisons (e.g., A vs. B and B vs. A) and self-comparisons (e.g., A vs. A) are removed to avoid redundancy. Missing genotype data is also accounted for and reported in the results.

Value

A data frame with the following columns:

- `parent1`: The name of the first parent in the pair.

- parent2: The name of the second parent in the pair.
- progeny: The name of the progeny sample.
- homoRef_x_homoRef_n: Number of loci where both parents are homozygous reference.
- homoRef_x_homoRef_match: Percentage of matching loci in the progeny for homozygous reference parents.
- homoAlt_x_homoAlt_n: Number of loci where both parents are homozygous alternate.
- homoAlt_x_homoAlt_match: Percentage of matching loci in the progeny for homozygous alternate parents.
- homoRef_x_homoAlt_n: Number of loci where one parent is homozygous reference and the other is homozygous alternate.
- homoRef_x_homoAlt_match: Percentage of matching loci in the progeny for mixed homozygous parents.
- homoalt_x_homoRef_n: Number of loci where one parent is homozygous alternate and the other is homozygous reference.
- homoalt_x_homoRef_match: Percentage of matching loci in the progeny for mixed homozygous parents (alternate-reference).
- missing: The number of loci with missing genotype data in the comparison.

Examples

```
# Example VCF file
example_vcf <- system.file("iris_DArT_VCF.vcf.gz", package = "BIGr")

parents_candidates <- paste0("Sample_",1:10)
progeny_candidates <- paste0("Sample_",11:20)

#Check homozygous loci in trios
check_tab <- check_homozygous_trios(path.vcf = example_vcf,
                                    ploidy = 2,
                                    parents_candidates = parents_candidates,
                                    progeny_candidates = progeny_candidates)
```

---

check_madc_sanity *Run basic sanity checks on a MADC-style allele report*

Description

Performs nine quick validations on an allele report:

1. **Columns** - required columns are present (CloneID, AlleleID, AlleleSequence);
2. **FixAlleleIDs** - first column's first up-to-6 rows are not all blank or "*" *and* both _0001 and _0002 appear in AlleleID;
3. **IUPACcodes** - presence of non-ATCG characters in AlleleSequence;

4. **LowerCase** - presence of lowercase a/t/c/g in `AlleleSequence`;
5. **Indels** - reference/alternate allele lengths differ for the same `CloneID`, or a `"-"` character is present in `AlleleSequence`;
6. **ChromPos** - all `CloneID` values follow the `Chr_Position` format (prefix matches `"chr"` case-insensitively, suffix is a positive integer);
7. **allNAcol** - at least one column contains only `NA` or empty values;
8. **allNArow** - at least one row contains only `NA` or empty values;
9. **RefAltSeqs** - every `CloneID` has at least one `Ref` and one `Alt` allele row;
10. **OtherAlleles** - presence of alleles where the target locus differs from both the Ref and Alt in `AlleleSequence`.

Usage

```
check_madc_sanity(report)
```

Arguments

`report`	A `data.frame` with at least the columns `CloneID`, `AlleleID`, and `AlleleSequence`. The first column is also used in the `FixAlleleIDs` check to inspect its first up to six entries. If `FixAlleleIDs` is `FALSE` (raw DArT format), the first 7 rows are treated as header filler and skipped before further checks are run.

Details

- **FixAlleleIDs:** When the first six rows of the first column are all blank or `"*"` (raw DArT format), row 7 is promoted to column headers and the first 7 rows are dropped before subsequent checks are run. The check is `TRUE` when the file has already been processed by HapApp (fixed IDs with `_0001`/`_0002` suffixes).
- **IUPAC check:** Flags any character outside `A`, `T`, `C`, `G` and `"-"` (case-insensitive), which includes ambiguity codes (`N`, `R`, `Y`, etc.).
- **Indels:** Rows are split by `AlleleID` containing `"Ref_0001"` vs `"Alt_0002"`, merged by `CloneID`, and flagged as indels if either (a) the lengths of `AlleleSequence` differ, (b) the sequences have the same length but more than one character differs between them (complex indel / local rearrangement), or (c) a `"-"` character is present anywhere in `AlleleSequence`.
- **ChromPos:** Each `CloneID` is split on `"_"` into exactly two parts; the first part must match `"Chr"` (case-insensitive) and the second must be a positive integer. Returns `FALSE` when any `CloneID` is `NA`.
- **allNAcol / allNArow:** Detected via `apply()` over columns/rows respectively; a cell is treated as missing when it is `NA` or an empty string (`""`). Useful for flagging empty or corrupt entries.
- **RefAltSeqs:** For each unique `CloneID`, checks whether at least one row with a `Ref` (`|Ref_` when `FixAlleleIDs = TRUE`, `|Ref$` otherwise) and one row with an `Alt` (`|Alt_` / `|Alt$`) allele exist. `CloneIDs` that lack a `Ref` row are stored in `missRef`; those lacking an `Alt` row in `missAlt`. The check is `TRUE` when both sets are empty.
- If required columns are missing (`Columns = FALSE`), only `Columns` and `FixAlleleIDs` are evaluated; all other checks remain `NA` and `indel_clone_ids`, `missRef`, and `missAlt` are returned as `NULL`.

Value

A named list with five elements:

checks Named logical vector with nine entries: `Columns`, `FixAlleleIDs`, `IUPACcodes`, `LowerCase`, `Indels`, `ChromPos`, `allNAcol`, `allNArow`, `RefAltSeqs`. `TRUE` means the condition was detected (or passed for `Columns`, `FixAlleleIDs`, `ChromPos`, and `RefAltSeqs`); `NA` means the check was skipped.

messages Named list of length-2 character vectors, one per check. Element `[[1]]` is the message when the check is `TRUE`, element `[[2]]` when it is `FALSE`. Indexed by the same names as `checks`.

indel_clone_ids Character vector of `CloneIDs` where ref/alt lengths differ. Returns `character(0)` if none are found, or `NULL` when required columns are missing.

missRef Character vector of `CloneIDs` that have no `Ref` allele row. Returns `character(0)` if all `CloneIDs` have a `Ref` row, or `NULL` when required columns are missing.

missAlt Character vector of `CloneIDs` that have no `Alt` allele row. Returns `character(0)` if all `CloneIDs` have an `Alt` row, or `NULL` when required columns are missing.

---

check_ped *Check a pedigree file for accuracy and report/correct common errors*

Description

`check_ped` reads a 3-column pedigree file (tab-separated, columns labeled `id`, `sire`, `dam` in any order) and performs quality checks, optionally correcting or flagging errors.

Usage

```
check_ped(ped.file, seed = NULL, verbose = TRUE)
```

Arguments

`ped.file`	Path to the pedigree text file.
`seed`	Optional seed for reproducibility.
`verbose`	Logical. If TRUE (default), prints the report to the console.

Details

The function checks for:

- Exact duplicate rows and removes them (keeping one copy)
- IDs that appear more than once with conflicting sire/dam assignments (sets sire/dam to "0")
- IDs that appear in both sire and dam columns
- Missing parents (IDs referenced as sire/dam but not in `id` column), adds them with sire/dam = "0"

- Direct and indirect pedigree dependencies (cycles), such as a parent being its own descendant

After an initial run to clean exact duplicates and repeated IDs, you can run the function again to detect cycles more accurately.

The function does **not** overwrite the input file or create objects in the global environment. Instead, it returns the report and corrected pedigree in a list.

Value

A list of data.frames containing detected issues:

- `exact_duplicates`: rows that were exact duplicates
- `repeated_ids_diff`: IDs appearing more than once with conflicting sire/dam
- `messy_parents`: IDs appearing as both sire and dam
- `missing_parents`: parents added to the pedigree with 0 as sire/dam
- `dependencies`: detected cycles in the pedigree
- `corrected_pedigree`: corrected pedigree table

Examples

```
ped_file <- system.file("check_ped_test.txt", package = "BIGr")
ped_errors <- check_ped(ped.file = ped_file, seed = 101919, verbose = FALSE)

# Access messy parents
ped_errors$messy_parents

# Access corrected pedigree
ped_errors$corrected_pedigree

# IDs with messy parents
messy_ids <- ped_errors$messy_parents$id
print(messy_ids)
```

---

check_replicates *Compatibility Between Samples Genotypes*

Description

This function checks the compatibility between sample genotypes in a VCF file by comparing all pairs of samples.

Usage

```
check_replicates(path.vcf, select_samples = NULL, verbose = TRUE)
```

Arguments

path.vcf	A string specifying the path to the VCF file containing genotype data.
select_samples	An optional character vector of sample names to be selected for comparison. If NULL (default), all samples in the VCF file are used.
verbose	A logical value indicating whether to print the number of combinations tested. Default is TRUE.

Details

The function removes reciprocal comparisons (e.g., A vs. B and B vs. A) and self-comparisons (e.g., A vs. A) to avoid redundancy. Compatibility is calculated as the percentage of matching genotypes between two samples, excluding missing values. The percentage of missing genotypes is also reported for each pair.

Value

A data frame with four columns:

- sample1: The name of the first sample in the pair.
- sample2: The name of the second sample in the pair.
- %_matching_genotypes: The percentage of compatible genotypes between the two samples.
- %_missing_genotypes: The percentage of missing genotypes in the comparison.

Examples

```
#Example VCF
example_vcf <- system.file("iris_DArT_VCF.vcf.gz", package = "BIGr")

# Checking for replicates
check_tab <- check_replicates(path.vcf = example_vcf, select_samples = NULL)
```

---

dosage_ratios *Calculate the Percentage of Each Dosage Value*

Description

This function calculates the percentage of each dosage value within a genotype matrix. It assumes that the samples are the columns, and the genomic markers are in rows. Missing data should be set as NA, which will then be ignored for the calculations. All samples must have the same ploidy.

Usage

```
dosage_ratios(data, ploidy)
```

Arguments

data	Genotype matrix or data.frame
ploidy	The ploidy of the species being analyzed

Value

A data.frame with percentages of dosage values in the genotype matrix

Examples

```
# example numeric genotype matrix for a tetraploid
n_ind <- 5
n_snps <- 10

geno <- matrix(as.numeric(sample(0:4, n_ind * n_snps, replace = TRUE)), nrow = n_snps, ncol = n_ind)
colnames(geno) <- paste0("Ind", 1:n_ind)
rownames(geno) <- paste0("SNP", 1:n_snps)
ploidy <- 4

# ratio of dosage value (numeric genotypes) across samples in dataset
result <- dosage_ratios(geno, ploidy)

print(result)
```

---

dosage2vcf *Convert DArTag genotype reports and counts to VCF*

Description

This function will convert DArT genotype report and Counts files to VCF format

Usage

```
dosage2vcf(dart.report, dart.counts, ploidy, output.file)
```

Arguments

dart.report	Path to the DArT Allele Dose Report or SNP/INDEL report .csv file.
dart.counts	Path to the DArT counts .csv file. Typically contains "Counts" in the file name.
ploidy	The ploidy of the species being analyzed
output.file	output file name and path

Details

This function will convert Allele Dose Report or SNP/INDEL report files and Counts files from DArT into a VCF file. These two files are received directly from DArT for a given sequencing project. SNP/INDEL one-row and two-row reports are treated as diploid genotype reports with 0 = reference homozygote, 1 = alternate homozygote, 2 = heterozygote, and - = missing. Allele Dose reports are interpreted as reference allele dosages using the supplied ploidy. The output file will be saved to the location and with the name that is specified. The VCF format is v4.3

Value

A vcf file

Examples

```
## Use file paths for each file on the local system

#The files are directly from DArT for a given sequencing project.
#The are labeled with Dosage_Report or Counts in the file names.

#Temp location (only for example)
output_file <- tempfile()

dosage2vcf(dart.report = system.file("iris_DArT_Allele_Dose_Report_small.csv", package = "BIGr"),
           dart.counts = system.file("iris_DArT_Counts_small.csv", package = "BIGr"),
           ploidy = 2,
           output.file = output_file)

# Removing the output for the example
rm(output_file)

##The function will output the converted VCF using information from the DArT files
```

`filterMADC` *Filter MADC Files*

Description

Filter and process MADC files to remove low quality microhaplotypes

Usage

```
filterMADC(
  madc_file,
  min.mean.reads = NULL,
  max.mean.reads = NULL,
  max.mhaps.per.loci = NULL,
  min.reads.per.site = 1,
```

```
  min.ind.with.reads = NULL,
  target.only = FALSE,
  n.summary.columns = NULL,
  output.file = NULL
)
```

Arguments

madc_file	Path to the MADC file to be filtered
min.mean.reads	Minimum mean read depth for filtering
max.mean.reads	Maximum mean read depth for filtering
max.mhaps.per.loci	Maximum number of matching mhaps per target loci. Retains only the target Ref and Alt loci at the sites that exceeds the max.mhaps.per.loci threshold.
min.reads.per.site	Minimum number of reads per site for min.ind.with.reads. Otherwise, this parameter is ignored
min.ind.with.reads	Minimum number of individuals with min.reads.per.site reads for filtering
target.only	Logical indicating whether to filter for target loci only
n.summary.columns	(optional) Number of summary columns to remove from MADC file not including the first three. Otherwise, the columns will be automatically detected and removed.
output.file	Path to save the filtered data (if NULL, data will not be saved)

Details

This function can filter raw MADC files or pre-processed MADC files with fixed allele IDs. Additionally, it can filter based on mean read depth, number of mhaps per target loci, and other criteria. Optionally, users can plot summary statistics and save the filtered data to a file.

Value

data.frame or saved csv file

Examples

```
#Example

#Example MADC
madc_file <- system.file("example_MADC_FixedAlleleID.csv", package="BIGr")

#Remove mhaps exceeding 3 per target region including the ref and alt target mhaps
filtered_df <- filterMADC(madc_file,
                          min.mean.reads = NULL,
                          max.mean.reads = NULL,
                          max.mhaps.per.loci = 3,
```

```
                    min.reads.per.site = 1,
                    min.ind.with.reads = NULL,
                    target.only = FALSE,
                    n.summary.columns = NULL,
                    output.file = NULL)
```

`filterVCF` *Filter a VCF file*

Description

This function will filter a VCF file or vcfR object and export the updated version

Usage

```
filterVCF(
  vcf.file,
  filter.OD = NULL,
  filter.BIAS.min = NULL,
  filter.BIAS.max = NULL,
  filter.DP = NULL,
  filter.MPP = NULL,
  filter.PMC = NULL,
  filter.MAF = NULL,
  filter.SAMPLE.miss = NULL,
  filter.SNP.miss = NULL,
  ploidy,
  output.file = NULL
)
```

Arguments

`vcf.file`	vcfR object or path to VCF file. Can be unzipped (.vcf) or gzipped (.vcf.gz).
`filter.OD`	Updog filter
`filter.BIAS.min`	Updog filter (requires a value for both BIAS.min and BIAS.max)
`filter.BIAS.max`	Updog filter (requires a value for both BIAS.min and BIAS.max)
`filter.DP`	Total read depth at each SNP filter
`filter.MPP`	Updog filter
`filter.PMC`	Updog filter
`filter.MAF`	Minor allele frequency filter

`filter.SAMPLE.miss`
: Sample missing data filter

`filter.SNP.miss`
: SNP missing data filter

`ploidy`
: The ploidy of the species being analyzed

`output.file`
: output file name (optional). If no output.file name provided, then a vcfR object will be returned.

Details

This function will input a VCF file or vcfR object and filter based on the user defined options. The output file will be saved to the location and with the name that is specified. The VCF format is v4.3

Value

A gzipped vcf file

Examples

```
## Use file paths for each file on the local system

#Temp location (only for example)
output_file <- tempfile()

filterVCF(vcf.file = system.file("iris_DArT_VCF.vcf.gz", package = "BIGr"),
           filter.OD = 0.5,
           filter.MAF = 0.05,
           ploidy = 2,
           output.file = output_file)

# Removing the output for the example
rm(output_file)

##The function will output the filtered VCF to the current working directory
```

---

find_parentage *Find Parentage Assignments for Progeny*

Description

Assigns the most likely parent(s) to each progeny individual based on genotypic data using Mendelian error rates or homozygous mismatch rates.

Usage

```
find_parentage(
  genotypes_file,
  parents_file,
  progeny_file,
  method = "best_pair",
  min_markers = 10,
  error_threshold = 5,
  show_ties = TRUE,
  allow_selfing = TRUE,
  verbose = TRUE,
  write_txt = TRUE
)
```

Arguments

genotypes_file	Path to a TSV/CSV/TXT file containing genotype data. Must include an 'ID' column followed by marker columns coded as 0, 1, 2 (allele dosage).
parents_file	Path to a TSV/CSV/TXT file listing candidate parent IDs. Must include an 'ID' column. An optional 'Sex' column with values 'M' (male parent), 'F' (female parent), or 'A' (ambiguous) determines which parents are tested for each role. If absent, all parents are treated as ambiguous.
progeny_file	Path to a TSV/CSV/TXT file listing progeny IDs to assign. Must include an 'ID' column.
method	Character. Parentage assignment method. One of: • "best_male_parent" - finds the best male parent for each progeny using homozygous mismatch rate. • "best_female_parent" - finds the best female parent for each progeny using homozygous mismatch rate. • "best_match" - finds the single best parent (either sex) using homozygous mismatch rate. • "best_pair" - finds the best male-female parent pair for each progeny using full Mendelian error rate (default).
min_markers	Integer. Minimum number of non-missing markers required to report a parentage assignment. Progeny-parent comparisons with fewer markers are flagged as LOW_MARKERS and no assignment is made (default: 10).
error_threshold	Numeric. Maximum mismatch percentage to report a parentage assignment as confident. Assignments above this threshold are flagged as HIGH_ERROR in the Assignment_Status column (default: 5.0). Must be between 0 and 100.
show_ties	Logical. If TRUE, all tied best pairs (after tie-breaking by maximum markers tested) are reported as additional columns (Male_Parent_1, Male_Parent_2, etc.) when method = "best_pair". The base columns (Male_Parent, Female_Parent, etc.) are always populated with the top result. If FALSE, only one tied pair is reported with a warning. Default is TRUE.

allow_selfing	Logical. If FALSE, male-female parent pairs where both IDs are identical are excluded when method = "best_pair". Default is TRUE.
verbose	Logical. If TRUE, prints progress messages, summary statistics, and the results table to the console. Default is TRUE.
write_txt	Logical. If TRUE, writes results to parentage_results_dt.txt in the working directory. Default is TRUE.

Details

A homozygous-only genotype matrix is pre-computed once at startup and shared across all methods that require it, avoiding redundant computation.

For "best_male_parent", "best_female_parent", and "best_match", only homozygous markers (coded 0 or 2) are used for comparison; heterozygous markers (coded 1) are set to NA. This reduces false mismatches caused by phase ambiguity.

For "best_pair", all markers are used and full Mendelian inheritance rules are applied. Mismatch rates and comparison counts are computed across all progeny simultaneously using vectorised vapply calls, producing n_pairs x n_progeny matrices and giving substantial speed gains for large datasets. Both matrices are explicitly coerced to matrix form to handle the edge case of a single parent pair correctly.

When multiple pairs share the lowest Mendelian error rate, ties are broken by selecting the pair(s) with the greatest number of markers tested. If ties still remain after this step, all remaining tied pairs are reported when show_ties = TRUE.

The base columns (Male_Parent, Female_Parent, Mendelian_Error_Pct, Markers_Tested) are always populated with the top result, ensuring no missing values in these columns regardless of tie status.

Output rows are pre-allocated as a data.table and filled by reference using data.table::set(), avoiding repeated memory allocation during the results-building step.

Individuals in parents_file or progeny_file that are absent from genotypes_file are removed with a warning.

Progeny with fewer non-missing markers than min_markers are flagged LOW_MARKERS and no parent assignment is reported. Progeny whose best match exceeds error_threshold are flagged HIGH_ERROR.

Value

A data.table with one row per progeny. Columns depend on the method used:

- best_male_parent / best_female_parent / best_match: Progeny, Best_Match, Mendelian_Error_Pct, Markers_Tested, Assignment_Status.
- best_pair (no ties after tie-breaking): Progeny, Male_Parent, Female_Parent, Mendelian_Error_Pct, Markers_Tested, Assignment_Status.
- best_pair (ties remain after tie-breaking, show_ties = TRUE): base columns are always populated with the top result, plus suffix columns Male_Parent_1, Female_Parent_1, etc. for each tied pair.

Assignment_Status is one of PASS, HIGH_ERROR, or LOW_MARKERS. Returned invisibly when verbose = TRUE.

Examples

```
## Not run:
# Assign best male-female parent pair to each progeny
results <- find_parentage(
  genotypes_file  = "genotypes.txt",
  parents_file    = "parents.txt",
  progeny_file    = "progeny.txt",
  method          = "best_pair",
  min_markers     = 50,
  error_threshold = 5.0,
  show_ties       = TRUE,
  allow_selfing   = FALSE
)

# Find best individual parent match (ignoring sex)
results <- find_parentage(
  genotypes_file  = "genotypes.txt",
  parents_file    = "parents.txt",
  progeny_file    = "progeny.txt",
  method          = "best_match",
  min_markers     = 30,
  error_threshold = 3.0
)

## End(Not run)
```

---

flip_dosage *Switch Dosage Values from a Genotype Matrix*

Description

This function converts the dosage count values to the opposite value. This is primarily used when converting dosage values from reference based (0 = homozygous reference) to alternate count based (0 = homozygous alternate). It assumes that the Samples are the columns, and the genomic markers are in rows. Missing data should be set as NA, which will then be ignored for the calculations. All samples must have the same ploidy.

Usage

```
flip_dosage(df, ploidy, is.reference = TRUE)
```

Arguments

df	Genotype matrix or data.frame
ploidy	The ploidy of the species being analyzed
is.reference	The dosage calls value is based on the count of reference alleles (TRUE/FALSE)

Value

A genotype matrix

Examples

```
# example code

# example numeric genotype matrix for a tetraploid
n_ind <- 5
n_snps <- 10

geno <- matrix(as.numeric(sample(0:4, n_ind * n_snps, replace = TRUE)), nrow = n_snps, ncol = n_ind)
colnames(geno) <- paste0("Ind", 1:n_ind)
rownames(geno) <- paste0("SNP", 1:n_snps)
ploidy <- 4

# Output matrix with the allele count reversed
results <- flip_dosage(geno, ploidy, is.reference = TRUE)

print(results)
```

---

get_countsMADC *Obtain Read Counts from MADC File*

Description

Reads a DArTag MADC report and returns reference and total read count matrices per marker and sample. Only `Ref` and `Alt` target loci are retained; `|AltMatch` / `|RefMatch` rows are either discarded or collapsed depending on `collapse_matches_counts`.

Usage

```
get_countsMADC(
  madc_file = NULL,
  madc_object = NULL,
  collapse_matches_counts = FALSE,
  verbose = TRUE
)
```

Arguments

`madc_file`	character or `NULL`. Path to the input MADC CSV file. At least one of `madc_file` or `madc_object` must be provided.
`madc_object`	data frame or `NULL`. A pre-read MADC data frame (e.g., as returned by `check_botloci()`). When supplied, file reading is skipped. At least one of `madc_file` or `madc_object` must be provided.

collapse_matches_counts
: logical. If TRUE, counts for |AltMatch and |RefMatch rows are summed into their corresponding |Ref and |Alt rows. If FALSE (default), |AltMatch and |RefMatch rows are discarded.

verbose
: logical. Whether to print progress messages. Default is TRUE.

Details

Either madc_file or madc_object must be provided (not both NULL). When madc_object is supplied it is passed directly to get_counts(), skipping file I/O. The function constructs:

- ref_matrix - per-sample reference allele counts.
- size_matrix - per-sample total counts (ref + alt).

Markers whose CloneID appears only in the Ref or only in the Alt rows are removed with a warning. A summary of the proportion of zero-count data points (missing data) is reported via vmsg().

Value

A named list with two numeric matrices, both with markers as rows and samples as columns:

ref_matrix Reference allele read counts.

size_matrix Total read counts (reference + alternative).

See Also

check_madc_sanity()

Examples

```
# Get the path to the MADC file
madc_path <- system.file("iris_DArT_MADC.csv", package = "BIGr")

# Extract the read count matrices
counts_matrices <- get_countsMADC(madc_path)

# Access the reference and size matrices
# ref_matrix  <- counts_matrices$ref_matrix
# size_matrix <- counts_matrices$size_matrix

rm(counts_matrices)
```

`imputation_concordance` *Calculate Concordance between Imputed and Reference Genotypes*

Description

This function calculates the concordance between imputed and reference genotypes. It assumes that samples are rows and markers are columns. Allele dosages (0, 1, 2) are recommended but other numeric formats are supported. Missing data in either dataset can be excluded from the concordance calculation using the `missing_code` argument. Specific markers can be excluded using the `snps_2_exclude` argument.

Usage

```
imputation_concordance(
  reference_genos,
  imputed_genos,
  missing_code = NULL,
  snps_2_exclude = NULL,
  verbose = FALSE,
  plot = FALSE,
  print_result = TRUE
)
```

Arguments

`reference_genos`	A data frame containing reference genotype data, with rows as samples and columns as markers. Must include a column named `ID`.
`imputed_genos`	A data frame containing imputed genotype data, with rows as samples and columns as markers. Must include a column named `ID`.
`missing_code`	Optional value specifying missing data. If provided, loci with this value in either dataset will be excluded from the concordance calculation.
`snps_2_exclude`	Optional vector of marker IDs to exclude from the concordance calculation.
`verbose`	Logical. If `TRUE`, prints summary statistics (minimum, quartiles, median, mean, maximum) of concordance percentages.
`plot`	Logical. If `TRUE`, produces a bar plot of concordance percentage by sample.
`print_result`	Logical. If `TRUE` (default), prints the concordance results data frame to the console. If `FALSE`, results are returned invisibly.

Details

The function:

1. Identifies common samples and markers between the datasets.

2. Optionally excludes specified SNPs.
3. Removes loci with missing data (if `missing_code` is provided).
4. Computes per-sample concordance as the percentage of matching genotypes.

When `plot = TRUE`, a bar plot showing concordance percentage per sample is generated using **ggplot2**.

Value

A data frame with:

- `ID`: Sample identifiers shared between the datasets.
- `Concordance`: Percentage of matching genotypes per sample.

If `print_result = FALSE`, the data frame is returned invisibly.

Examples

```
ref <- data.frame(
  ID = c("S1", "S2", "S3"),
  SNP1 = c(0, 1, 2),
  SNP2 = c(1, 1, 0),
  SNP3 = c(2, 5, 1)
)

test <- data.frame(
  ID = c("S1", "S2", "S3"),
  SNP1 = c(0, 0, 2),
  SNP2 = c(1, 1, 1),
  SNP3 = c(2, 5, 0)
)

result <- imputation_concordance(
  reference_genos = ref,
  imputed_genos = test,
  snps_2_exclude = "SNP2",
  missing_code = 5,
  print_result = FALSE
)

result
```

`madc2gmat` *Convert MADC Files to an Additive Genomic Relationship Matrix*

Description

Scale and normalize MADC read count data and convert it to an additive genomic relationship matrix.

Usage

```
madc2gmat(
  madc_file,
  seed = NULL,
  method = "collapsed",
  ploidy,
  output.file = NULL
)
```

Arguments

madc_file	Path to the MADC file to be filtered
seed	Optional seed for random number generation (default is NULL)
method	Method to use for processing the MADC data. Options are "unique" or "collapsed". Default is "collapsed".
ploidy	Numeric. Ploidy level of the samples (e.g., 2 for diploid, 4 for tetraploid)
output.file	Path to save the filtered data (if NULL, data will not be saved)

Details

This function reads a MADC file, processes it to remove unnecessary columns, and then converts it into an additive genomic relationship matrix using the first method proposed by VanRaden (2008). The resulting matrix can be used for genomic selection or other genetic analyses.

Value

data.frame or saved csv file

Author(s)

Alexander M. Sandercock

References

VanRaden, P. M. (2008). Efficient methods to compute genomic predictions. Journal of Dairy Science, 91(11), 4414-4423

Examples

```
#Input variables
madc_file <- system.file("example_MADC_FixedAlleleID.csv", package="BIGr")

#Calculations
temp <- tempfile()

# Converting to additive relationship matrix
gmat <- madc2gmat(madc_file,
                  seed = 123,
                  ploidy=2,
```

```
                    output.file = NULL)
```

`madc2vcf_all` *Converts MADC file to VCF recovering target and off-target SNPs*

Description

This function processes a MADC file to generate a VCF file containing both target and off-target SNPs. It includes options for filtering multiallelic SNPs and parallel processing to improve performance.

Usage

```
madc2vcf_all(
  madc,
  botloci_file,
  hap_seq_file = NULL,
  n.cores = 1,
  rm_multiallelic_SNP = FALSE,
  multiallelic_SNP_dp_thr = 0,
  multiallelic_SNP_sample_thr = 0,
  alignment_score_thr = 40,
  out_vcf = NULL,
  markers_info = NULL,
  add_others = TRUE,
  others_max_snps = 5,
  others_rm_with_indels = TRUE,
  verbose = TRUE
)
```

Arguments

`madc`	Required. A string specifying the path or URL to the MADC file.
`botloci_file`	Required. A string specifying the path or URL to the file containing the target IDs designed in the bottom strand.
`hap_seq_file`	A string specifying the path to the haplotype database fasta file.
`n.cores`	An integer specifying the number of cores to use for parallel processing. Default is 1.
`rm_multiallelic_SNP`	A logical value. If TRUE, SNPs with more than one alternative base are removed. If FALSE, the thresholds specified by `multiallelic_SNP_dp_thr` and `multiallelic_SNP_sample_thr` are used to filter low-frequency SNP alleles. Default is FALSE.

`multiallelic_SNP_dp_thr`
: A numeric value specifying the minimum depth by tag threshold for filtering low-frequency SNP alleles when `rm_multiallelic_SNP` is FALSE. Default is 0.

`multiallelic_SNP_sample_thr`
: A numeric value specifying the minimum number of samples threshold for filtering low-frequency SNP alleles when `rm_multiallelic_SNP` is FALSE. Default is 0.

`alignment_score_thr`
: A numeric value specifying the minimum alignment score threshold. Default is 40.

`out_vcf`
: A string specifying the name of the output VCF file. If the file extension is not `.vcf`, it will be appended automatically.

`markers_info`
: A string specifying the path to a CSV file with marker information (CloneID/BI_markerID, Chr, Pos, Ref, Alt, Type, Indel_pos columns as needed).

`add_others`
: A logical value. If TRUE, alleles labeled "Other" in the MADC file are included in off-target SNP extraction. Default is TRUE.

`others_max_snps`
: An integer or NULL. If not NULL, Other alleles with more than this many SNP differences versus the Ref sequence (as detected by pairwise alignment) are discarded. Default is NULL (no limit).

`others_rm_with_indels`
: A logical value. If TRUE, Other alleles that contain insertions or deletions relative to the Ref sequence (as detected by pairwise alignment) are discarded. Default is TRUE.

`verbose`
: A logical value indicating whether to print metrics and progress to the console. Default is TRUE.

Details

The function processes a MADC file to generate a VCF file containing both target and off-target SNPs. It uses parallel processing to improve performance and provides options to filter multiallelic SNPs based on user-defined thresholds. The alignment score threshold can be adjusted using the `alignment_score_thr` parameter. The generated VCF file includes metadata about the processing parameters and the BIGr package version. If the `alignment_score_thr` is not met, the corresponding SNPs are discarded.

Sanity check behaviour and requirements

Check	Status	Required
Indels	detected	`markers_info` with `Ref`/`Alt`/`Indel_pos`/`Indel_length` + `botloci_file`
	not detected	`botloci_file`
ChromPos	valid	`botloci_file`
	invalid	`markers_info` with `Chr`/`Pos` + `botloci_file`
RefAltSeqs	all paired	`botloci_file`
	unpaired	`botloci_file` + `hap_seq_file` (microhaplotype DB)

Value

This function does not return an R object. It writes the processed VCF file v4.3 to the specified `out_vcf` path.

Examples

```
# Example usage:


Sys.setenv("OMP_THREAD_LIMIT" = 2)

madc_file <- system.file("example_MADC_FixedAlleleID.csv", package="BIGr")
bot_file <- system.file("example_SNPs_DArTag-probe-design_f180bp.botloci", package="BIGr")
db_file <- system.file("example_allele_db.fa", package="BIGr")

#Temp location (only for example)
output_file <- tempfile()

madc2vcf_all(
  madc = madc_file,
  botloci_file = bot_file,
  hap_seq_file = db_file,
  n.cores = 2,
  rm_multiallelic_SNP = TRUE,
  multiallelic_SNP_dp_thr = 10,
  multiallelic_SNP_sample_thr = 5,
  alignment_score_thr = 40,
  out_vcf = output_file,
  verbose = TRUE
)

rm(output_file)
```

`madc2vcf_multi` *Convert MADC file to VCF using polyRAD for multiallelic genotyping*

Description

This function converts a DArTag fixed allele ID MADC file to a VCF containing multiallelic markers based on the microhaplotypes using the polyRAD package's `readDArTag`, `IterateHWE` population model and `RADdata2VCF` pipeline.

Usage

```
madc2vcf_multi(
  madc_file,
  botloci_file,
```

```
  outfile,
  markers_info = NULL,
  ploidy = 2L,
  verbose = TRUE
)
```

Arguments

madc_file	character. Path or URL to the input MADC CSV file.
botloci_file	character. Path or URL to the botloci file listing target IDs designed on the bottom strand.
outfile	character. Path for the output VCF file.
markers_info	character or NULL. Optional path or URL to a CSV file with marker metadata. Required when CloneIDs do not follow the Chr_Pos format; must contain CloneID (or BI_markerID), Chr, and Pos columns.
ploidy	integer. Ploidy level of the samples passed to taxaPloidy. Default is 2.
verbose	logical. Whether to print progress messages. Default is TRUE.

Details

The function performs the following steps:

1. Reads the MADC file and runs check_madc_sanity.
2. Validates the botloci file against MADC CloneIDs using check_botloci, fixing any padding mismatches automatically.
3. Converts lowercase bases in AlleleSequence to uppercase if detected.
4. Removes all-NA rows and columns if detected.
5. Writes the corrected data to a temporary file and passes it to polyRAD::readDArTag.
6. Estimates overdispersion with polyRAD::TestOverdispersion and calls polyRAD::IterateHWE, then exports the result with polyRAD::RADdata2VCF.

Sanity check behaviour and requirements

The function always stops if IUPAC codes, unpaired Ref/Alt sequences, or unfixed AlleleIDs are detected (see check_madc_sanity). For the remaining checks the required inputs are:

Check	Status	Required
Indels	detected	botloci_file
	not detected	botloci_file
ChromPos	valid	botloci_file
	invalid	markers_info with Chr/Pos + botloci_file

Value

Invisible NULL. Writes a VCF file to outfile.

madc2vcf_targets *Format MADC Target Loci Read Counts Into VCF*

Description

Parses a DArTag **MADC** report and writes a **VCF v4.3** containing per-target read counts for the panel's target loci. This is useful because MADC is not widely supported by general-purpose tools, while VCF is.

Usage

```
madc2vcf_targets(
  madc_file,
  output.file,
  botloci_file = NULL,
  markers_info = NULL,
  get_REF_ALT = FALSE,
  collapse_matches_counts = FALSE,
  verbose = TRUE
)
```

Arguments

madc_file	character. Path to the input MADC CSV file.
output.file	character. Path to the output VCF file to write.
botloci_file	character or NULL (default NULL). Path to a plain-text file listing target IDs designed on the **bottom** strand (one ID per line). Used for strand-correcting probe sequences when get_REF_ALT = TRUE and markers_info does not supply Ref and Alt columns. Not needed when markers_info provides Ref and Alt, or when get_REF_ALT = FALSE and markers_info provides Chr and Pos. Also required when ChromPos is invalid and markers_info does not provide Ref/Alt.
markers_info	character or NULL. Optional path to a CSV providing target metadata. Matching is done by column name, not column position. Accepted columns: • one marker identifier column named CloneID, Marker_ID, or BI_markerID (required; a generic ID column is not accepted); • Chr, Pos - required when CloneID does not follow the Chr_Pos format; • Ref, Alt - required when get_REF_ALT = TRUE and probe-sequence inference is not possible (IUPAC codes, indels, or unfixed allele IDs). When get_REF_ALT = TRUE, botloci_file is still required unless Ref and Alt are supplied here.
get_REF_ALT	logical (default FALSE). If TRUE, attempts to recover REF/ALT bases. The source is chosen automatically: markers_info Ref/Alt columns take priority; otherwise probe sequences from the MADC are compared (with botloci_file for strand correction). Targets with more than one difference between Ref/Alt sequences are removed. When FALSE, REF and ALT are set to "." in the output VCF.

`collapse_matches_counts`
: logical (default `FALSE`). If `TRUE`, counts for `|AltMatch` and `|RefMatch` rows are summed into their corresponding `|Ref` and `|Alt` rows before building the matrices. Useful when the MADC contains multiple allele rows per target that should be aggregated.

`verbose`
: logical (default `TRUE`). If `TRUE`, prints detailed progress messages about each processing step.

Details

Convert DArTag MADC target read counts to a VCF

What this function does

- Runs basic sanity checks on the MADC file via `check_madc_sanity()` (column presence, fixed allele IDs, IUPAC/ambiguous bases, lowercase bases, indels, chromosome/position format, all-NA rows/columns, Ref/Alt sequence presence).
- Extracts reference and total read counts per sample and target.
- Derives `AD` (ref,alt) by subtraction (alt = total - ref).
- If `get_REF_ALT = TRUE`, recovers REF/ALT bases either from `markers_info` (when `Ref`/`Alt` columns are present) or by comparing the Ref/Alt probe sequences in the MADC file (with strand correction via `botloci_file`). Targets with >1 polymorphism between sequences are discarded.
- Optionally accepts a `markers_info` CSV to supply `CHROM`, `POS`, `REF`, `ALT`, bypassing sequence-based inference.

Output VCF layout

- `INFO` fields:
  - `DP` - total depth across all samples for the locus
  - `ADS` - total counts across samples in the order `ref,alt`
- `FORMAT` fields (per sample):
  - `DP` - total reads (ref + alt)
  - `RA` - reads supporting the reference allele
  - `AD` - `"ref,alt"` counts

Strand handling If a target ID appears in `botloci_file`, its probe sequences are reverse- complemented prior to base comparison so that REF/ALT are reported in the top-strand genomic orientation.

Sanity check behaviour and requirements

The function always stops if required columns (`CloneID`, `AlleleID`, `AlleleSequence`) are missing.

For the remaining checks the required inputs depend on the combination of check result and `get_REF_ALT`:

Check	Status	`get_REF_ALT`	Required
IUPAC codes	detected	`TRUE`	`markers_info` with `Ref`/`Alt`
	detected	`FALSE`	none
	not detected	`TRUE`	`botloci_file` **or** `markers_info` with `Ref`/`Alt`

	not detected	FALSE	none
Indels	detected	TRUE	markers_info with Ref/Alt
	detected	FALSE	none
	not detected	TRUE	botloci_file **or** markers_info with Ref/Alt
	not detected	FALSE	none
ChromPos	valid	TRUE	botloci_file **or** markers_info with Ref/Alt
	valid	FALSE	none
	invalid	TRUE	markers_info with Chr/Pos/Ref/Alt **or** markers_info with Chr/Pos + botl
	invalid	FALSE	markers_info with Chr/Pos
FixAlleleIDs	fixed	TRUE	botloci_file **or** markers_info with Ref/Alt
	fixed	FALSE	none
	not fixed	TRUE	markers_info with Ref/Alt
	not fixed	FALSE	none

Value

(Invisibly) returns the path to output.file. The side effect is a **VCF v4.3** written to disk containing one row per target and columns for all samples in the MADC file.

Dependencies

Uses **dplyr**, **tidyr**, **tibble**, **reshape2**, **Biostrings** and base **utils**. Helper functions expected in this package: check_madc_sanity(), get_countsMADC(), get_counts(), and check_botloci().

See Also

check_madc_sanity(), get_countsMADC(), check_botloci()

Examples

```
# Example files shipped with the package
madc_file <- system.file("example_MADC_FixedAlleleID.csv", package = "BIGr")
bot_file  <- system.file("example_SNPs_DArTag-probe-design_f180bp.botloci",
                         package = "BIGr")
out_vcf <- tempfile(fileext = ".vcf")

# Convert MADC to VCF (attempting to recover REF/ALT from probe sequences)
## Not run:
madc2vcf_targets(
  madc_file    = madc_file,
  output.file  = out_vcf,
  botloci_file = bot_file,
  get_REF_ALT  = TRUE
)

## End(Not run)

# Clean up (example)
unlink(out_vcf)
```

merge_MADCs *Merge MADC files*

Description

If duplicated samples exist in different files, a suffix will be added at the end of the sample name. If run_ids is defined, they are used as suffix, if not, files will be identified from 1 to number of files, considering the order that was defined in the function.

Usage

```
merge_MADCs(..., madc_list = NULL, out_madc = NULL, run_ids = NULL)
```

Arguments

...	one or more MADC files path
madc_list	list containing path to MADC files to be merged
out_madc	output merged MADC file path
run_ids	vector of character defining the run ID for each file. This ID will be added as a suffix in repeated sample ID in case they exist in different files.

Value

A data frame containing the merged MADC data. The merged file is also written to the specified out_madc path in CSV format. Numeric columns are filled with zeros where data is missing.

Examples

```
# First generating example MADC files
temp_dir <- tempdir()
file1_path <- file.path(temp_dir, "madc1.csv")
file2_path <- file.path(temp_dir, "madc2.csv")
out_path <- file.path(temp_dir, "merged_madc.csv")

# Data for file 1: Has SampleA and SampleB
df1 <- data.frame(
 AlleleID = c("chr1.1_0001|Alt_0002", "chr1.1_0001|Ref_0001", "chr1.1_0001|AltMatch_0001"),
  CloneID = c("chr1.1_0001", "chr1.1_0001", "chr1.1_0001"),
  AlleleSequence = c("GGG", "AAA", "TTT"),
  SampleA = c(10, 8, 0),
  SampleB = c(5, 4, 9),
  stringsAsFactors = FALSE,
  check.names = FALSE
)
write.csv(df1, file1_path, row.names = FALSE, quote = FALSE)

# Data for file 2: Has SampleA (duplicate name) and SampleC, different rows
df2 <- data.frame(
```

```
 AlleleID = c("chr1.1_0001|Alt_0002", "chr1.1_0001|Ref_0001", "chr1.1_0001|AltMatch_0001"),
  CloneID = c("chr1.1_0001", "chr1.1_0001", "chr1.1_0001"),
  AlleleSequence = c("GGG", "AAA", "TTT"),
  SampleA = c(11, 7, 20),
  SampleC = c(1, 2, 6),
  stringsAsFactors = FALSE,
  check.names = FALSE
)
write.csv(df2, file2_path, row.names = FALSE, quote = FALSE)

# 2. Run the merge function
# Use default suffixes (.x, .y) for the duplicated "SampleA"
merge_MADCs(madc_list = list(file1_path, file2_path),
            out_madc = out_path)
```

`solve_composition_poly` *Compute Genome-Wide Breed Composition*

Description

Computes genome-wide breed/ancestry composition using quadratic programming on a batch of animals.

Usage

```
solve_composition_poly(
  Y,
  X,
  ped = NULL,
  groups = NULL,
  mia = FALSE,
  sire = FALSE,
  dam = FALSE,
  ploidy = 2
)
```

Arguments

`Y`	numeric matrix of genotypes (columns) from all animals (rows) in population coded as dosage of allele B `{0, 1, 2, ..., ploidy}`
`X`	numeric matrix of allele frequencies (rows) from each reference panel (columns). Frequencies are relative to allele B.
`ped`	data.frame giving pedigree information. Must be formatted "ID", "Sire", "Dam"
`groups`	list of IDs categorized by breed/population. If specified, output will be a list of results categorized by breed/population.

mia	logical. Only applies if ped argument is supplied. If true, returns a data.frame containing the inferred maternally inherited allele for each locus for each animal instead of breed composition results.
sire	logical. Only applies if ped argument is supplied. If true, returns a data.frame containing sire genotypes for each locus for each animal instead of breed composition results.
dam	logical. Only applies if ped argument is supplied. If true, returns a data.frame containing dam genotypes for each locus for each animal instead of breed composition results.
ploidy	integer. The ploidy level of the species (e.g., 2 for diploid, 3 for triploid, etc.).

Value

A data.frame or list of data.frames (if groups is !NULL) with breed/ancestry composition results

References

Funkhouser SA, Bates RO, Ernst CW, Newcom D, Steibel JP. Estimation of genome-wide and locus-specific breed composition in pigs. Transl Anim Sci. 2017 Feb 1;1(1):36-44.

Examples

```
# Example inputs for solve_composition_poly (ploidy = 4)

# (This would typically be the output from allele_freq_poly)
allele_freqs_matrix <- matrix(
  c(0.625, 0.500,
    0.500, 0.500,
    0.500, 0.500,
    0.750, 0.500,
    0.625, 0.625),
  nrow = 5, ncol = 2, byrow = TRUE,
  dimnames = list(paste0("SNP", 1:5), c("VarA", "VarB"))
)

# Validation Genotypes (individuals x SNPs)
val_geno_matrix <- matrix(
  c(2, 1, 2, 3, 4,  # Test1 dosages for SNP1-5
    3, 4, 2, 3, 0), # Test2 dosages for SNP1-5
  nrow = 2, ncol = 5, byrow = TRUE,
  dimnames = list(paste0("Test", 1:2), paste0("SNP", 1:5))
)

# Calculate Breed Composition
composition <- solve_composition_poly(Y = val_geno_matrix,
                                      X = allele_freqs_matrix,
                                      ploidy = 4)
print(composition)
```

thinSNP *Thin a dataframe of SNPs based on genomic position*

Description

This function groups SNPs by chromosome, sorts them by physical position, and then iteratively selects SNPs such that no two selected SNPs within the same chromosome are closer than a specified minimum distance.

Usage

```
thinSNP(df, chrom_col_name, pos_col_name, min_distance)
```

Arguments

df	The input dataframe.
chrom_col_name	A string specifying the name of the chromosome column.
pos_col_name	A string specifying the name of the physical position column.
min_distance	A numeric value for the minimum distance between selected SNPs. The unit of this distance should match the unit of the pos_col_name column (e.g., base pairs).

Value

A thinned dataframe with the same columns as the input.

Examples

```
# Create sample SNP data
set.seed(123)
n_snps <- 20
snp_data <- data.frame(
  MarkerID = paste0("SNP", 1:n_snps),
  Chrom = sample(c("chr1", "chr2"), n_snps, replace = TRUE),
  ChromPosPhysical = c(
    sort(sample(1:1000, 5)), # SNPs on chr1
    sort(sample(1:1000, 5)) + 500, # More SNPs on chr1
    sort(sample(1:2000, 10))      # SNPs on chr2
  ),
  Allele = sample(c("A/T", "G/C"), n_snps, replace = TRUE)
)
# Ensure it's sorted by Chrom and ChromPosPhysical for clarity in example
snp_data <- snp_data[order(snp_data$Chrom, snp_data$ChromPosPhysical), ]
rownames(snp_data) <- NULL

print("Original SNP data:")
print(snp_data)
```

```
# Thin the SNPs, keeping a minimum distance of 100 units (e.g., bp)
thinned_snps <- thinSNP(
  df = snp_data,
  chrom_col_name = "Chrom",
  pos_col_name = "ChromPosPhysical",
  min_distance = 100
)

print("Thinned SNP data (min_distance = 100):")
print(thinned_snps)

# Thin with a larger distance
thinned_snps_large_dist <- thinSNP(
  df = snp_data,
  chrom_col_name = "Chrom",
  pos_col_name = "ChromPosPhysical",
  min_distance = 500
)
print("Thinned SNP data (min_distance = 500):")
print(thinned_snps_large_dist)
```

`updog2vcf` *Export Updog Results as VCF*

Description

This function will convert an Updog output to a VCF file

Usage

```
updog2vcf(
  multidog.object,
  output.file,
  updog_version = NULL,
  RefAlt = NULL,
  compress = TRUE
)
```

Arguments

`multidog.object`	updog output object with class "multidog" from dosage calling
`output.file`	output file name and path
`updog_version`	character defining updog package version used to generate the multidog object
`RefAlt`	optional data frame with four columns named "Chr", "Pos", "Ref", and "Alt" containing the reference and alternate alleles for each SNP in the same order as in the multidog object
`compress`	logical. If TRUE returns a vcf.gz file

Details

When performing dosage calling for multiple SNPs using Updog, the output file contains information for all loci and all samples. This function will convert the updog output file to a VCF file, while retaining the information for the values that are commonly used to filter low quality and low confident dosage calls.

Value

A vcf file

References

Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping polyploids from messy sequencing data. Genetics, 210(3), 789-807.

Examples

```
# Retrieving the updog output multidog object
load(system.file("extdata", "iris-multidog.rdata", package = "BIGr"))

temp_file <- tempfile()

# Convert updog to VCF, where the new VCF will be saved at the location specified in the output.file
updog2vcf(
  multidog.object = mout,
  output.file = temp_file,
  updog_version = "0.0.0",
  compress = TRUE
)

#Removing the example vcf
rm(temp_file)
```

---

validate_pedigree *Validate Pedigree Trios Using Mendelian Error Analysis*

Description

Validates parent-offspring trios by calculating Mendelian error rates from SNP genotype data. Identifies incorrect parentage assignments and suggests best-matching replacements. If a list of founders is supplied, trios that are declared founders (both parents coded as 0) are preserved unchanged with no recommendations. Trios removed due to missing genotype data are retained in the output with a NO_GENOTYPE_DATA status.

Usage

```
validate_pedigree(
  pedigree_file,
  genotypes_file,
  founders_file = NULL,
  trio_error_threshold = 5,
  min_markers = 10,
  single_parent_error_threshold = 2,
  verbose = TRUE,
  write_txt = TRUE,
  output_filename = "pedigree_validation_results.txt"
)
```

Arguments

pedigree_file	Character. Path to the pedigree file (TSV/CSV/TXT) with columns: ID, Male_Parent, Female_Parent.
genotypes_file	Character. Path to the genotypes file (TSV/CSV/TXT) with an ID column followed by marker columns coded as 0, 1, 2.
founders_file	Character, optional. Path to a one-column file listing the IDs of founder individuals. Founders with both parents coded as 0 are left unchanged with no recommendations. Defaults to NULL.
trio_error_threshold	Numeric. Maximum Mendelian error percentage to classify a trio as PASS (default: 5.0). Must be between 0 and 100.
min_markers	Integer. Minimum number of non-missing markers required to evaluate a trio (default: 10).
single_parent_error_threshold	Numeric. Maximum homozygous-marker mismatch percentage for a parent to be considered acceptable during parent-level evaluation (default: 2.0). Must be between 0 and 100.
verbose	Logical. If TRUE, prints progress messages, a summary table, and results to the console (default: TRUE).
write_txt	Logical. If TRUE, writes validation results to output_filename (default: TRUE).
output_filename	Character. Path/name of the output file (default: "pedigree_validation_results.txt").

Value

A data.table (returned invisibly) with one row per trio and the following columns:

ID Individual ID.

Male_Parent Declared male parent ID.

Female_Parent Declared female parent ID.

Mendelian_Error_Pct Trio-level Mendelian error percentage.

Markers_Tested Number of markers with non-missing genotypes.

Status One of PASS, FAIL, LOW_MARKERS, NO_DATA, FOUNDERS, MISSING_MALE_PARENT, MISSING_FEMALE_PARENT, MISSING_BOTH_PARENTS, or NO_GENOTYPE_DATA.

Correction_Decision One of NONE, KEEP_BOTH, REMOVE_MALE_PARENT, REMOVE_FEMALE_PARENT, REMOVE_BOTH, LOW_MARKERS_KEEP_BOTH, LOW_MARKERS_REMOVE_MALE_PARENT, LOW_MARKERS_REMOVE_FEMALE_PARENT, LOW_MARKERS_REMOVE_BOTH.

Male_Parent_Hom_Error_Pct Male parent homozygous-marker mismatch percentage.

Female_Parent_Hom_Error_Pct Female parent homozygous-marker mismatch percentage.

Best_Male_Parent Best-matching male parent candidate ID.

Best_Male_Parent_Error_Pct Homozygous mismatch percentage for the best male parent candidate.

Best_Female_Parent Best-matching female parent candidate ID.

Best_Female_Parent_Error_Pct Homozygous mismatch percentage for the best female parent candidate.

Index