Package ‘BIGr’

May 19, 2025

Title Breeding Insight Genomics Functions for Polyploid and Diploid Species

Version 0.5.5

Maintainer Alexander M. Sandercock <ams866@cornell.edu>

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

RoxygenNote 7.3.2

Depends R (>= 4.4.0)

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

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

RdMacros Rdpack

NeedsCompilation no

Author Alexander M. Sandercock [cre, aut],
Cristiane Taniguti [aut],
Josue Chinchilla-Vargas [aut],
Shufen Chen [ctb],
Manoj Sapkota [ctb],
Meng Lin [ctb],
Dongyan Zhao [ctb],
Cornell University [cph] (Breeding Insight)

Repository CRAN

Date/Publication 2025-05-19 09:00:06 UTC

Contents

calculate_Het 2
calculate_MAF 3
check_homozygous_trios 4
check_ped 6
check_replicates 7
dosage2vcf 8
dosage_ratios 9
filterVCF 10
flip_dosage 11
get_countsMADC 12
imputation_concordance 13
madc2vcf_all 14
madc2vcf_targets 16
merge_MADCs 17
updog2vcf 19

Index **21**

`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_ped *Evaluate Pedigree File for Accuracy*

Description

Check a pedigree file for accuracy and output suspected errors

Usage

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

Arguments

ped.file	path to pedigree text file. The pedigree file is a 3-column pedigree tab separated file with columns labeled as id sire dam in any order
seed	Optional seed for reproducibility
verbose	Logical. If TRUE, print the errors to the console.

Details

check_ped takes a 3-column pedigree tab separated file with columns labeled as id sire dam in any order and checks for:

- Ids that appear more than once in the id column
- Ids that appear in both sire and dam columns
- Direct (e.g. parent is a offspring of his own daughter) and indirect (e.g. a great grandparent is son of its grandchild) dependencies within the pedigree.
- Individuals included in the pedigree as sire or dam but not on the id column and reports them back with unknown parents (0).

When using check_ped, do a first run to check for repeated ids and parents that appear as sire and dam. Once these errors are cleaned run the function again to check for dependencies as this will provide the most accurate report.

Note: This function does not change the input file but prints any errors found in the console.

Value

A list of data.frames of error types, and the output printed to the console

Examples

```
##Get list with a dataframe for each error type
ped_file <- system.file("check_ped_test.txt", package="BIGr")
ped_errors <- check_ped(ped.file = ped_file,
                        seed = 101919)

##Access the "messy parents" dataframe result
```

```
ped_errors$messy_parents

##Get list of sample IDs with messy parents error
messy_parent_ids <- ped_errors$messy_parents$id
print(messy_parent_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)
```

---

dosage2vcf *Convert DArTag Dosage and Counts to VCF*

Description

This function will convert the DArT Dosage Report and Counts files to VCF format

Usage

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

Arguments

dart.report	Path to the DArT dosage report .csv file. Typically contains "Dosage Report" in the file name.
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 the Dosage Report and Counts files from DArT into a VCF file. These two files are received directly from DArT for a given sequencing project. 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
```

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)
```

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
```

`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

This function takes the MADC file as input and retrieves the ref and alt counts for each sample, and converts them to ref, alt, and size(total count) matrices for dosage calling tools. At the moment, only the read counts for the Ref and Alt target loci are obtained while the additional loci are ignored.

Usage

```
get_countsMADC(madc_file)
```

Arguments

madc_file	Path to MADC file

Value

A list of read count matrices for reference, alternate, and total read count values

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, alternate, and size matrices

# ref_matrix <- counts_matrices$ref_matrix
# alt_matrix <- counts_matrices$alt_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. It is recommended to use allele dosages (0, 1, 2) but will work with other formats. Missing data in reference or imputed genotypes will not be considered for concordance if the `missing_code` argument is used. If a specific subset of markers should be excluded, it can be provided using the `snps_2_exclude` argument.

Usage

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

Arguments

`reference_genos`
: A data frame containing reference genotype data, with rows as samples and columns as markers. Dosage format (0, 1, 2) is recommended.

`imputed_genos`
: A data frame containing imputed genotype data, with rows as samples and columns as markers. Dosage format (0, 1, 2) is recommended.

`missing_code`
: An optional value to specify missing data. If provided, loci with this value in either dataset will be excluded from the concordance calculation.

`snps_2_exclude`
: An optional vector of marker IDs to exclude from the concordance calculation.

`verbose` A logical value indicating whether to print a summary of the concordance results. Default is FALSE.

Details

The function identifies common samples and markers between the reference and imputed genotype datasets. It calculates the percentage of matching genotypes for each sample, excluding missing data and specified markers. The concordance is reported as a percentage for each sample, along with a summary of the overall concordance distribution.

Value

A list with two elements:

- `result_df`: A data frame with sample IDs and their concordance percentages.
- `summary_concordance`: A summary of concordance percentages, including minimum, maximum, mean, and quartiles.

Examples

```
# Example Input variables
ignore_file <- system.file("imputation_ignore.txt", package="BIGr")
ref_file <- system.file("imputation_reference.txt", package="BIGr")
test_file <- system.file("imputation_test.txt", package="BIGr")

# Import files
snps = read.table(ignore_file, header = TRUE)
ref = read.table(ref_file, header = TRUE)
test = read.table(test_file, header = TRUE)

#Calculations
result <- imputation_concordance(reference_genos = ref,
                                 imputed_genos = test,
                                 snps_2_exclude = snps,
                                 missing_code = 5,
                                 verbose = FALSE)
```

`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 = NULL,
  botloci_file = NULL,
  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,
  verbose = TRUE
)
```

Arguments

`madc` A string specifying the path to the MADC file.

`botloci_file` A string specifying the path 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.

`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.

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_targets` *Format MADC Target Loci Read Counts Into VCF*

Description

This function will extract the read count information from a MADC file target markers and convert to VCF file format.

Usage

```
madc2vcf_targets(madc_file, output.file, botloci_file, get_REF_ALT = FALSE)
```

Arguments

`madc_file`	Path to MADC file
`output.file`	output file name and path
`botloci_file`	A string specifying the path to the file containing the target IDs designed in the bottom strand.
`get_REF_ALT`	if TRUE recovers the reference and alternative bases by comparing the sequences. If more than one polymorphism are found for a tag, it is discarded.

Details

The DArTag MADC file format is not commonly supported through existing tools. This function will extract the read count information from a MADC file for the target markers and convert it to a VCF file format for the genotyping panel target markers only

Value

A VCF file v4.3 with the target marker read count information

A VCF file v4.3 with the target marker read count information

Examples

```
# Load example files
madc_file <- system.file("example_MADC_FixedAlleleID.csv", package="BIGr")
bot_file <- system.file("example_SNPs_DArTag-probe-design_f180bp.botloci", package="BIGr")

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

# Convert MADC to VCF
madc2vcf_targets(madc_file = madc_file,
                 output.file = output_file,
                 get_REF_ALT = TRUE,
                 botloci_file = bot_file)

rm(output_file)
```

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)
```

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, 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
`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)
```

Index

calculate_Het, 2
calculate_MAF, 3
check_homozygous_trios, 4
check_ped, 6
check_replicates, 7

dosage2vcf, 8
dosage_ratios, 9

filterVCF, 10
flip_dosage, 11

get_countsMADC, 12

imputation_concordance, 13

madc2vcf_all, 14
madc2vcf_targets, 16
merge_MADCs, 17

updog2vcf, 19