Package: genio (via r-universe)

August 24, 2024

Title Genetics Input/Output Functions

Version 1.1.4.9000

Description Implements readers and writers for file formats associated with genetics data. Reading and writing Plink BED/BIM/FAM and GCTA binary GRM formats is fully supported, including a lightning-fast BED reader and writer implementations. Other functions are 'readr' wrappers that are more constrained, user-friendly, and efficient for these particular applications; handles Plink and Eigenstrat tables (FAM, BIM, IND, and SNP files). There are also make functions for FAM and BIM tables with default values to go with simulated genotype data.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports readr (>= 2.0.0), tibble, dplyr, Rcpp, R.utils

LinkingTo Rcpp

Suggests testthat, knitr, rmarkdown, BEDMatrix, snpStats, lobstr

VignetteBuilder knitr

URL https://github.com/OchoaLab/genio

BugReports https://github.com/OchoaLab/genio/issues

Repository https://ochoalab.r-universe.dev

RemoteUrl https://github.com/ochoalab/genio

RemoteRef HEAD

RemoteSha c5fa573ea5d39db82d23ead18424c54d95142fb1

Contents

`count_lines` *Count the number of lines of a file*

Description

This function returns the number of lines in a file. It is intended to result in fast retrieval of numbers of individuals (from FAM or equivalent files) or loci (BIM or equivalent files) when the input files are extremely large and no other information is required from these files. This code uses C++ to quickly counts lines (like linux's `wc -l` but this one is cross-platform).

Usage

```
count_lines(file, ext = NA, verbose = TRUE)
```

Arguments

file	The input file path to read (a string).
ext	An optional extension. If NA (default), file is expected to exist as-is. Otherwise, if file doesn't exist and the extension was missing, then this extension is added.
verbose	If TRUE (default), writes a message reporting the file whose lines are being counted (after adding extensions if it was needed).

Details

Note: this function does not work correctly with compressed files (they are not uncompressed prior to counting newlines).

Value

The number of lines in the file.

Examples

```
# count number of individuals from an existing plink *.fam file
file <- system.file("extdata", 'sample.fam', package = "genio", mustWork = TRUE)
n_ind <- count_lines(file)
n_ind

# count number of loci from an existing plink *.bim file
file <- system.file("extdata", 'sample.bim', package = "genio", mustWork = TRUE)
m_loci <- count_lines(file)
m_loci
```

---

delete_files_grm *Delete all GCTA binary GRM files*

Description

This function deletes each of the GCTA binary GRM files (grm.bin, grm.N.bin, and grm.id extensions) given the shared base file path, warning if any of the files did not exist or if any were not successfully deleted.

Usage

```
delete_files_grm(file)
```

Arguments

file	The shared file path (excluding extensions: grm.bin, grm.N.bin, or grm.id).

Value

Nothing

Examples

```
# if you want to delete "data.grm.bin", "data.grm.N.bin" and "data.grm.id", run like this:
# delete_files_grm("data")

# The following example is more awkward
# because (only for these examples) the package must create *temporary* files to actually delete

# create dummy GRM files
file <- tempfile('delete-me-test') # no extension
# add each extension and create empty files
file.create( paste0(file, '.grm.bin') )
file.create( paste0(file, '.grm.N.bin') )
file.create( paste0(file, '.grm.id') )

# delete the GRM files we just created
delete_files_grm(file)
```

---

delete_files_phen *Delete PHEN files*

Description

This function deletes a PHEN files given the base file path (without extension), warning if the file did not exist or if it was not successfully deleted.

Usage

```
delete_files_phen(file)
```

Arguments

file	The base file path (excluding phen extension).

Value

Nothing

Examples

```
# if you want to delete "data.phen", run like this:
# delete_files_phen("data")

# The following example is more awkward
# because (only for these examples) the package must create a *temporary* file to actually delete

# create dummy PHEN files
file <- tempfile('delete-me-test') # no extension
# add extension and create an empty file
file.create( paste0(file, '.phen') )

# delete the PHEN file we just created
delete_files_phen(file)
```

---

delete_files_plink *Delete all Plink binary files*

Description

This function deletes each of the Plink binary files (`bed`, `bim`, `fam` extensions) given the shared base file path, warning if any of the files did not exist or if any were not successfully deleted.

Usage

```
delete_files_plink(file)
```

Arguments

`file`	The shared file path (excluding extensions: `bed`, `bim`, `fam`).

Value

Nothing

Examples

```
# if you want to delete "data.bed", "data.bim" and "data.fam", run like this:
# delete_files_plink("data")

# The following example is more awkward
# because (only for these examples) the package must create *temporary* files to actually delete

# create dummy BED/BIM/FAM files
file <- tempfile('delete-me-test') # no extension
# add each extension and create empty files
file.create( paste0(file, '.bed') )
file.create( paste0(file, '.bim') )
```

```
file.create( paste0(file, '.fam') )

# delete the BED/BIM/FAM files we just created
delete_files_plink(file)
```

---

genio *genio (GENetics I/O): A package for reading and writing genetics data*

Description

This package fully supports reading and writing Plink BED/BIM/FAM and GCTA GRM files, as illustrated below. These functions make it easy to create dummy annotation tables to go with simulated genotype data too. Lastly, there is functionality to read and write Eigenstrat tables.

Author(s)

Maintainer: Alejandro Ochoa <alejandro.ochoa@duke.edu> (ORCID)

See Also

Useful links:

- https://github.com/OchoaLab/genio
- Report bugs at https://github.com/OchoaLab/genio/issues

Examples

```
# read existing BED/BIM/FAM files

# first get path to BED file
file <- system.file( "extdata", 'sample.bed', package = "genio", mustWork = TRUE )

# read genotypes and annotation tables
plink_data <- read_plink( file )
# genotypes
X <- plink_data$X
# locus annotations
bim <- plink_data$bim
# individual annotations
fam <- plink_data$fam

# the same works without .bed extension
file <- sub( '\\.bed$', '', file ) # remove extension
plink_data <- read_plink( file )

# write data into new BED/BIM/FAM files
file_out <- tempfile( 'delete-me-example' )
write_plink( file_out, X, bim, fam )
```

```
# delete example files when done
delete_files_plink( file_out )

# read sample GRM files
file <- system.file( "extdata", 'sample.grm.bin', package = "genio", mustWork = TRUE )
file <- sub( '\\.grm\\.bin$', '', file ) # remove extension from this path on purpose
obj <- read_grm( file )
# the kinship matrix
kinship <- obj$kinship
# the pair sample sizes matrix
M <- obj$M
# the fam and ID tibble
fam <- obj$fam

# write data into new GRM files
write_grm( file_out, kinship, M = M, fam = fam )

# delete example files when done
delete_files_grm( file_out )

# other functions not shown here allow reading and writing individual files,
# creating dummy tables to go with simulated genotypes,
# requiring the existence of these files,
# and reading and writing of Eigenstrat tables too.
```

`geno_to_char` *Convert a genotype matrix from numeric to character codes*

Description

Given the genotype matrix `X` and `bim` table (as they are parsed by `read_plink()`, this outputs a matrix of the same dimensions as `X` but with the numeric codes (all values in 0, 1, 2) translated to human-readable character codes (such as 'A/A', 'A/G', 'G/G', depending on which are the two alleles at the locus as given in the `bim` table, see return value).

Usage

```
geno_to_char(X, bim)
```

Arguments

`X`	The genotype matrix. It must have values only in 0, 1, 2, and `NA`.
`bim`	The variant table. It is required to have the same number of rows as `X`, and to have at least two named columns `alt` and `ref` (alleles 1 and 2 in a plink BIM table). These alleles can be arbitrary strings (i.e. not just SNPs but also indels, any single or multicharacter code, or even blank strings) except the forward slash character ("/") is not allowed anywhere in these strings (function stops if a slash is present), since in the output it is the delimiter string. `ref` and `alt` alleles must be different at each locus.

Value

The genotype matrix reencoded as strings. At one locus, if the two alleles (alt and ref) are 'A' and 'B', then the genotypes in the input are encoded as characters as: 0 -> 'A/A', 1 -> 'B/A', and 2 -> 'B/B'. Thus, the numeric encoding counts the reference allele dosage. `NA` values in input `X` remain `NA` in the output. If the input genotype matrix had row and column names, these are inherited by the output matrix.

See Also

`read_plink()`, `read_bed()`, `read_bim()`.

Examples

```
# a numeric/dosage genotype matrix with two loci (rows)
# and three individuals (columns)
X <- rbind( 0:2, c(0, NA, 2) )
# corresponding variant table (minimal case with just two required columns)
library(tibble)
bim <- tibble( alt = c('C', 'GT'), ref = c('A', 'G') )

# genotype matrix translated as characters
X_char <- geno_to_char( X, bim )
X_char
```

---

het_reencode_bed *Reencode a Plink BED file to (twice) heterozygote indicators*

Description

Given an existing plink-formatted BED (binary) file, this function reads it, transforms genotypes on the go, and writes a new BED file such that heterozygotes are encoded as 2 and homozygotes as 0. In other words, it transforms the numerical genotype values `c( 0, 1, 2, NA )` into `c( 0, 2, 0, NA )`. Heterozygotes are encoded as 2, rather than 1, so existing code for calculating allele frequencies and related quantities, such as kinship estimates, works on this data as intended. Intended to transform extremely large files that should not be loaded entirely into memory at once.

Usage

```
het_reencode_bed(
  file_in,
  file_out,
  m_loci = NA,
  n_ind = NA,
  make_bim_fam = TRUE,
  verbose = TRUE
)
```

Arguments

file_in	Input file path. *.bed extension may be omitted (will be added automatically if file doesn't exist but file.bed does).
file_out	Output file path. *.bed extension may be omitted (will be added automatically if it is missing).
m_loci	Number of loci in the input genotype table. If NA, it is deduced from the paired *.bim file
n_ind	Number of individuals in the input genotype table. If NA, it is deduced from the paired *.fam file
make_bim_fam	If TRUE, create symbolic links (using symlink()) for the output file's *.bim and *.fam that link to the corresponding input files. Otherwise only the *.bed file is created.
verbose	If TRUE (default) function reports the path of the files being read and written to (after autocompleting the extension).

See Also

read_bed() and write_bed(), from which much of the code of this function is derived, which explains additional BED format requirements.

Examples

```
# define input and output, both of which will also work without extension
# read an existing Plink *.bed file
file_in <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)
# write to a *temporary* location for this example
file_out <- tempfile('delete-me-example')

# in default mode, deduces dimensions from paired *.bim and *.fam tables
het_reencode_bed( file_in, file_out )

# delete output when done
delete_files_plink( file_out )
```

---

ind_to_fam *Convert an Eigenstrat IND tibble into a Plink FAM tibble*

Description

This function takes an existing IND tibble and creates a FAM tibble with the same information and dummy values for missing data. In particular, the output FAM tibble will contain these columns with these contents (IND only contain id, sex, and label, so there is no loss of information):

- fam: IND label
- id: IND id

- pat: 0 (missing paternal ID)
- mat: 0 (missing maternal ID)
- sex: IND sex converted to Plink integer codes via sex_to_int()
- peno: 0 (missing phenotype)

Usage

```
ind_to_fam(ind)
```

Arguments

ind	The input Eigenstrat IND tibble to convert.

Value

A Plink FAM tibble.

See Also

sex_to_int()

Eigenstrat IND format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

Examples

```
# create a sample IND tibble
library(tibble)
ind <- tibble(
  id = 1:3,
  sex = c('U', 'M', 'F'),
  label = c(1, 1, 2)
)
# convert to FAM
fam <- ind_to_fam(ind)
# inspect:
fam
```

---

make_bim *Create a Plink BIM tibble*

Description

This function simplifies the creation of Plink BIM-formatted tibbles, which autocompletes missing information if a partial tibble is provided, or generates a completely made up tibble if the number of individuals is provided. The default values are most useful for simulated genotypes, where IDs can be made up but must be unique, and there are no chromosomes, positions, or particular reference or alternative alleles.

Usage

```
make_bim(tib, n = NA)
```

Arguments

tib	The input tibble (optional). Missing columns will be autocompleted with reasonable values that are accepted by Plink and other external software. If missing, all will be autocompleted, but n is required.
n	The desired number of loci (rows). Required if tib is missing; otherwise it is ignored.

Details

Autocompleted column values:

- chr: 1 (all data is on a single chromosome)
- id: 1:n
- posg: 0 (missing)
- pos: 1:n
- ref: 1
- alt: 2

Note that n is either given directly or obtained from the input tibble.

Value

The input tibble with autocompleted columns and columns in default order, or the made up tibble if only the number of individuals was provided. The output begins with the standard columns in standard order: chr, id, posg, pos, ref, alt. Additional columns in the input tibble are preserved but placed after the standard columns.

See Also

Plink BIM format reference: https://www.cog-genomics.org/plink/1.9/formats#bim

Examples

```
# create a synthetic tibble for 10 loci
# (most common use case)
bim <- make_bim(n = 10)

# manually create a partial tibble with only chromosomes defined
library(tibble)
bim <- tibble(chr = 0:2)
# autocomplete the rest of the columns
bim <- make_bim(bim)
```

make_fam *Create a Plink FAM tibble*

Description

This function simplifies the creation of Plink FAM-formatted tibbles, which autocompletes missing information if a partial tibble is provided, or generates a completely made up tibble if the number of individuals is provided. The default values are most useful for simulated genotypes, where IDs can be made up but must be unique, and there are no parents, families, gender, or phenotype.

Usage

```
make_fam(tib, n = NA)
```

Arguments

`tib`	The input tibble (optional). Missing columns will be autocompleted with reasonable values that are accepted by Plink and other external software. If missing, all will be autocompleted, but `n` is required.
`n`	The desired number of individuals (rows). Required if `tib` is missing; otherwise it is ignored.

Details

Autocompleted column values:

- `fam`: `1:n`
- `id`: `1:n`
- `pat`: `0` (missing)
- `mat`: `0` (missing)
- `sex`: `0` (missing)
- `pheno`: `0` (missing)

Note that `n` is either given directly or obtained from the input tibble.

Value

The input tibble with autocompleted columns and columns in default order, or the made up tibble if only the number of individuals was provided. The output begins with the standard columns in standard order: `fam`, `id`, `pat`, `mat`, `sex`, `pheno`. Additional columns in the input tibble are preserved but placed after the standard columns.

See Also

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

Examples

```
# create a synthetic tibble for 10 individuals
# (most common use case)
fam <- make_fam(n = 10)

# manually create a partial tibble with only phenotypes defined
library(tibble)
fam <- tibble(pheno = 0:2)
# autocomplete the rest of the columns
fam <- make_fam(fam)
```

`read_bed` *Read a genotype matrix in Plink BED format*

Description

This function reads genotypes encoded in a Plink-formatted BED (binary) file, returning them in a standard R matrix containing genotypes encoded numerically as dosages (values in `c( 0, 1, 2, NA )`). Each genotype per locus (`m` loci) and individual (`n` total) counts the number of reference alleles, or `NA` for missing data. No *.fam or *.bim files are read by this basic function. Since BED does not encode the data dimensions internally, these values must be provided by the user.

Usage

```
read_bed(
  file,
  names_loci = NULL,
  names_ind = NULL,
  m_loci = NA,
  n_ind = NA,
  ext = "bed",
  verbose = TRUE
)
```

Arguments

`file`	Input file path. *.bed extension may be omitted (will be added automatically if `file` doesn't exist but `file`.bed does). See `ext` option below.
`names_loci`	Vector of loci names, to become the row names of the genotype matrix. If provided, its length sets `m_loci` below. If `NULL`, the returned genotype matrix will not have row names, and `m_loci` must be provided.
`names_ind`	Vector of individual names, to become the column names of the genotype matrix. If provided, its length sets `n_ind` below. If `NULL`, the returned genotype matrix will not have column names, and `n_ind` must be provided.

m_loci	Number of loci in the input genotype table. Required if names_loci = NULL, as its value is not deducible from the BED file itself. Ignored if names_loci is provided.
n_ind	Number of individuals in the input genotype table. Required if names_ind = NULL, as its value is not deducible from the BED file itself. Ignored if names_ind is provided.
ext	The desired file extension (default "bed"). Ignored if file points to an existing file. Set to NA to force file to exist as-is.
verbose	If TRUE (default) function reports the path of the file being read (after autocompleting the extension).

Details

The code enforces several checks to validate data given the requested dimensions. Errors are thrown if file terminates too early or does not terminate after genotype matrix is filled. In addition, as each locus is encoded in an integer number of bytes, and each byte contains up to four individuals, bytes with fewer than four are padded. To agree with other software (plink2, BEDMatrix), byte padding values are ignored (may take on any value without causing errors).

This function only supports locus-major BED files, which are the standard for modern data. Format is validated via the BED file's magic numbers (first three bytes of file). Older BED files can be converted using Plink.

Value

The m-by-n genotype matrix.

See Also

read_plink() for reading a set of BED/BIM/FAM files.

geno_to_char() for translating numerical genotypes into more human-readable character encodings.

Plink BED format reference: https://www.cog-genomics.org/plink/1.9/formats#bed

Examples

```
# first obtain data dimensions from BIM and FAM files
# all file paths
file_bed <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)
file_bim <- system.file("extdata", 'sample.bim', package = "genio", mustWork = TRUE)
file_fam <- system.file("extdata", 'sample.fam', package = "genio", mustWork = TRUE)
# read annotation tables
bim <- read_bim(file_bim)
fam <- read_fam(file_fam)

# read an existing Plink *.bim file
# pass locus and individual IDs as vectors, setting data dimensions too
X <- read_bed(file_bed, bim$id, fam$id)
X
```

```
# can specify without extension
file_bed <- sub('\\.bed$', '', file_bed) # remove extension from this path on purpose
file_bed # verify .bed is missing
X <- read_bed(file_bed, bim$id, fam$id) # loads too!
X
```

`read_bim` *Read Plink *.bim files*

Description

This function reads a standard Plink *.bim file into a tibble with named columns. It uses `readr::read_table()` to do it efficiently.

Usage

```
read_bim(file, verbose = TRUE)
```

Arguments

`file`	Input file (whatever is accepted by `readr::read_table()`). If file as given does not exist and is missing the expected *.bim extension, the function adds the .bim extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.bim extension is added as needed) is still not found and did not already contain the .gz extension and adding it points to an existing file.
`verbose`	If `TRUE` (default) function reports the path of the file being loaded (after autocompleting the extensions).

Value

A tibble with columns: `chr`, `id`, `posg`, `pos`, `alt`, `ref`.

See Also

`read_plink()` for reading a set of BED/BIM/FAM files.

Plink BIM format references: https://www.cog-genomics.org/plink/1.9/formats#bim https://www.cog-genomics.org/plink/2.0/formats#bim

Examples

```
# to read "data.bim", run like this:
# bim <- read_bim("data")
# this also works
# bim <- read_bim("data.bim")

# The following example is more awkward
```

```
# because package sample data has to be specified in this weird way:

# read an existing Plink *.bim file
file <- system.file("extdata", 'sample.bim', package = "genio", mustWork = TRUE)
bim <- read_bim(file)
bim

# can specify without extension
file <- sub('\\.bim$', '', file) # remove extension from this path on purpose
file # verify .bim is missing
bim <- read_bim(file) # loads too!
bim
```

`read_eigenvec` *Read Plink eigenvec file*

Description

This function reads a Plink eigenvec file, parsing columns strictly. First two must be 'fam' and 'id', which are strings, and all remaining columns (eigenvectors) must be numeric.

Usage

```
read_eigenvec(
  file,
  ext = "eigenvec",
  plink2 = FALSE,
  comment = if (plink2) "" else "#",
  verbose = TRUE
)
```

Arguments

`file`	The input file path, potentially excluding extension.
`ext`	File extension (default "eigenvec") can be changed if desired. Set to `NA` to force `file` to exist as-is.
`plink2`	If `TRUE`, the header is parsed and preserved in the returned data. The first two columns must be FID and IID, which are mandatory.
`comment`	A string used to identify comments. Any text after the comment characters will be silently ignored. Passed to `readr::read_table()`. '#' (default when `plink2 = FALSE`) works for Plink 2 eigenvec files, which have a header lines that starts with this character (the header is therefore ignored). However, `plink2 = TRUE` forces the header to be parsed instead.
`verbose`	If `TRUE` (default) function reports the path of the file being written (after autocompleting the extension).

Value

A list with two elements:

- `eigenvec`: A numeric R matrix containing the parsed eigenvectors. If `plink2 = TRUE`, the original column names will be preserved in this matrix.
- `fam`: A tibble with two columns, `fam` and `id`, which are the first two columns of the parsed file. These column names are always the same even if `plink2 = TRUE` (i.e. they won't be `FID` or `IID`).

See Also

`write_eigenvec()` for writing an eigenvec file.

Plink 1 eigenvec format reference: https://www.cog-genomics.org/plink/1.9/formats#eigenvec

Plink 2 eigenvec format reference: https://www.cog-genomics.org/plink/2.0/formats#eigenvec

GCTA eigenvec format reference: https://cnsgenomics.com/software/gcta/#PCA

Examples

```
# to read "data.eigenvec", run like this:
# data <- read_eigenvec("data")
# this also works
# data <- read_eigenvec("data.eigenvec")
#
# either way you get a list with these two items:
# numeric eigenvector matrix
# data$eigenvec
# fam/id tibble
# data$fam

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# read an existing *.eigenvec file created by GCTA
file <- system.file("extdata", 'sample-gcta.eigenvec', package = "genio", mustWork = TRUE)
data <- read_eigenvec(file)
# numeric eigenvector matrix
data$eigenvec
# fam/id tibble
data$fam

# can specify without extension
file <- sub('\\.eigenvec$', '', file) # remove extension from this path on purpose
file # verify .eigenvec is missing
data <- read_eigenvec(file) # load it anyway!
data$eigenvec

# read an existing *.eigenvec file created by Plink 2
file <- system.file("extdata", 'sample-plink2.eigenvec', package = "genio", mustWork = TRUE)
# this version ignores header
data <- read_eigenvec(file)
```

```
# numeric eigenvector matrix
data$eigenvec
# fam/id tibble
data$fam

# this version uses header
data <- read_eigenvec(file, plink2 = TRUE)
# numeric eigenvector matrix
data$eigenvec
# fam/id tibble
data$fam
```

`read_fam` *Read Plink *.fam files*

Description

This function reads a standard Plink *.fam file into a tibble with named columns. It uses `readr::read_table()` to do it efficiently.

Usage

```
read_fam(file, verbose = TRUE)
```

Arguments

`file`	Input file (whatever is accepted by `readr::read_table()`). If file as given does not exist and is missing the expected *.fam extension, the function adds the .fam extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.fam extension is added as needed) is still not found and did not already contain the .gz extension and adding it points to an existing file.
`verbose`	If `TRUE` (default) function reports the path of the file being loaded (after autocompleting the extensions).

Value

A tibble with columns: `fam`, `id`, `pat`, `mat`, `sex`, `pheno`.

See Also

`read_plink()` for reading a set of BED/BIM/FAM files.

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

Examples

```
# to read "data.fam", run like this:
# fam <- read_fam("data")
# this also works
# fam <- read_fam("data.fam")

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# read an existing Plink *.fam file
file <- system.file("extdata", 'sample.fam', package = "genio", mustWork = TRUE)
fam <- read_fam(file)
fam

# can specify without extension
file <- sub('\\.fam$', '', file) # remove extension from this path on purpose
file # verify .fam is missing
fam <- read_fam(file) # load it anyway!
fam
```

---

read_grm *Read GCTA GRM and related plink2 binary files*

Description

This function reads a GCTA Genetic Relatedness Matrix (GRM, i.e. kinship) set of files in their binary format, returning the kinship matrix and, if available, the corresponding matrix of pair sample sizes (non-trivial under missingness) and individuals table. Setting some options allows reading plink2 binary kinship formats such as "king" (see examples).

Usage

```
read_grm(
  name,
  n_ind = NA,
  verbose = TRUE,
  ext = "grm",
  shape = c("triangle", "strict_triangle", "square"),
  size_bytes = 4,
  comment = "#"
)
```

Arguments

name	The base name of the input files. Files with that base, plus shared extension (default "grm", see `ext` below), plus extensions `.bin`, `.N.bin`, and `.id` are read

if they exist. Only .<ext>.bin is absolutely required; .<ext>.id can be substituted by the number of individuals (see below); .<ext>.N.bin is entirely optional.

n_ind — The number of individuals, required if the file with the extension .<ext>.id is missing. If the file with the .<ext>.id extension is present, then this n_ind is ignored.

verbose — If TRUE (default), function reports the path of the files being loaded.

ext — Shared extension for all three inputs (see name above; default "grm"). Another useful value is "king" for KING-robust estimates produced by plink2. If NA, no extension is added. If given ext is also present at the end of name, then it is not added again.

shape — The shape of the information to read (may be abbreviated). Default "triangle" assumes there are n*(n+1)/2 values to read corresponding to the upper triangle including the diagonal (required for GCTA GRM). "strict_triangle" assumes there are n*(n-1)/2 values to read corresponding to the upper triangle *excluding* the diagonal (best for plink2 KING-robust). Lastly, "square" assumes there are n*n values to read corresponding to the entire square matrix, ignoring symmetry.

size_bytes — The number of bytes per number encoded. Default 4 corresponds to GCTA GRM and plink2 "bin4", whereas plink2 "bin" requires a value of 8.

comment — Character to start comments in <ext>.id file only. Default "#" helps plink2 .id files (which have a header that starts with "#", which is therefore ignored) be read just like plink1 and GCTA files (which do not have a header).

Value

A list with named elements:

- kinship: The symmetric n-times-n kinship matrix (GRM). Has IDs as row and column names if the file with extension .<ext>.id exists. If shape='strict_triangle', diagonal will have missing values.
- M: The symmetric n-times-n matrix of pair sample sizes (number of non-missing loci pairs), if the file with extension .<ext>.N.bin exists. Has IDs as row and column names if the file with extension .<ext>.id was available. If shape='strict_triangle', diagonal will have missing values.
- fam: A tibble with two columns: fam and id, same as in Plink FAM files. Returned if the file with extension .<ext>.id exists.

See Also

write_grm()

Greatly adapted from sample code from GCTA: https://cnsgenomics.com/software/gcta/#MakingaGRM

Examples

```
# to read "data.grm.bin" and etc, run like this:
# obj <- read_grm("data")
# obj$kinship # the kinship matrix
# obj$M       # the pair sample sizes matrix
# obj$fam     # the fam and ID tibble

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# read an existing set of GRM files
file <- system.file("extdata", 'sample.grm.bin', package = "genio", mustWork = TRUE)
file <- sub('\\.grm\\.bin$', '', file) # remove extension from this path on purpose
obj <- read_grm(file)
obj$kinship # the kinship matrix
obj$M       # the pair sample sizes matrix
obj$fam     # the fam and ID tibble

# Read sample plink2 KING-robust files (several variants).
# Read both base.king.bin and base.king.id files.
# All generated with "plink2 <input> --make-king <options> --out base"
# (replace "base" with actual base name) with these options:
# #1) "triangle bin"
# data <- read_grm( 'base', ext = 'king', shape = 'strict', size_bytes = 8 )
# #2) "triangle bin4"
# data <- read_grm( 'base', ext = 'king', shape = 'strict' )
# #3) "square bin"
# data <- read_grm( 'base', ext = 'king', shape = 'square', size_bytes = 8 )
# #4) "square bin4"
# data <- read_grm( 'base', ext = 'king', shape = 'square' )
```

---

read_ind *Read Eigenstrat *.ind files*

Description

This function reads a standard Eigenstrat *.ind file into a tibble. It uses `readr::read_table()` to do it efficiently.

Usage

```
read_ind(file, verbose = TRUE)
```

Arguments

`file`	Input file (whatever is accepted by `readr::read_table()`). If file as given does not exist and is missing the expected *.ind extension, the function adds the .ind extension and uses that path if that file exists. Additionally, the .gz extension is

added automatically if the file (after *.ind extension is added as needed) is still not found and did not already contain the .gz extension and adding it points to an existing file.

verbose — If TRUE (default), function reports the path of the file being loaded (after autocompleting the extensions).

Value

A tibble with columns: id, sex, label.

See Also

Eigenstrat IND format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

Examples

```
# to read "data.ind", run like this:
# ind <- read_ind("data")
# this also works
# ind <- read_ind("data.ind")

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# read an existing Eigenstrat *.ind file
file <- system.file("extdata", 'sample.ind', package = "genio", mustWork = TRUE)
ind <- read_ind(file)
ind

# can specify without extension
file <- sub('\\.ind$', '', file) # remove extension from this path on purpose
file # verify .ind is missing
ind <- read_ind(file) # load it anyway!
ind
```

---

read_matrix *Read a numerical matrix file into an R matrix*

Description

Reads a matrix file under strict assumptions that it is entirely numeric and there are no row or column names present in this file. It uses readr::read_table() to do it efficiently. Intended for outputs such as those of admixture inference approaches.

Usage

```
read_matrix(file, ext = "txt", verbose = TRUE)
```

Arguments

file	Input file (whatever is accepted by readr::read_table()). If file as given does not exist and is missing the expected extension (see ext below), the function adds the extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after the extension is added as needed) is still not found and did not already contain the .gz extension and adding it points to an existing file.
ext	The desired file extension. Ignored if file points to an existing file. Set to NA to force file to exist as-is.
verbose	If TRUE (default) function reports the path of the file being loaded (after autocompleting the extensions).

Value

A numeric matrix without row or column names.

See Also

write_matrix(), the inverse function.

Examples

```
# to read "data.txt", run like this:
# mat <- read_matrix("data")
# this also works
# mat <- read_matrix("data.txt")

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# read an existing matrix *.txt file
file <- system.file("extdata", 'sample-Q3.txt', package = "genio", mustWork = TRUE)
mat <- read_matrix(file)
mat

# can specify without extension
file <- sub('\\.txt$', '', file) # remove extension from this path on purpose
file # verify .txt is missing
mat <- read_matrix(file) # load it anyway!
mat
```

---

read_phen *Read *.phen files*

Description

This function reads a standard *.phen file into a tibble. It uses readr::read_table() to do it efficiently. GCTA and EMMAX use this format.

Usage

```
read_phen(file, verbose = TRUE)
```

Arguments

file	Input file (whatever is accepted by readr::read_table()). If file as given does not exist and is missing the expected *.phen extension, the function adds the .phen extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.phen extension is added as needed) is still not found and did not already contain the .gz extension and adding it points to an existing file.
verbose	If TRUE (default), function reports the path of the file being loaded (after autocompleting the extensions).

Value

A tibble with columns: fam, id, pheno.

See Also

GCTA PHEN format reference: https://cnsgenomics.com/software/gcta/#GREMLanalysis

Examples

```
# to read "data.phen", run like this:
# phen <- read_phen("data")
# this also works
# phen <- read_phen("data.phen")

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# read an existing plink *.phen file
file <- system.file("extdata", 'sample.phen', package = "genio", mustWork = TRUE)
phen <- read_phen(file)
phen

# can specify without extension
file <- sub('\\.phen$', '', file) # remove extension from this path on purpose
file # verify .phen is missing
phen <- read_phen(file) # load it anyway!
phen
```

read_plink *Read genotype and sample data in a Plink BED/BIM/FAM file set.*

Description

This function reads a genotype matrix (`X`, encoded as reference allele dosages) and its associated locus (`bim`) and individual (`fam`) data tables in the three Plink files in BED, BIM, and FAM formats, respectively. All inputs must exist or an error is thrown. This function is a wrapper around the more basic functions `read_bed()`, `read_bim()`, `read_fam()`, which simplifies data parsing and additionally better guarantees data integrity. Below suppose there are `m` loci and `n` individuals.

Usage

```
read_plink(file, verbose = TRUE)
```

Arguments

`file`	Input file path, without extensions (each of .bed, .bim, .fam extensions will be added automatically as needed). Alternatively, input file path may have .bed extension (but not .bim, .fam, or other extensions).
`verbose`	If `TRUE` (default), function reports the paths of the files being read (after autocompleting the extensions).

Value

A named list with items in this order: `X` (genotype matrix, see description in return value of `read_bed()`), `bim` (tibble, see `read_bim()`), `fam` (tibble, see `read_fam()`). `X` has row and column names corresponding to the `id` values of the `bim` and `fam` tibbles.

See Also

`read_bed()`, `read_bim()`, and `read_fam()` for individual parsers of each input table, including a description of each object returned.

`geno_to_char()` for translating numerical genotypes into more human-readable character encodings.

Plink BED/BIM/FAM format reference: https://www.cog-genomics.org/plink/1.9/formats

Examples

```
# to read "data.bed" etc, run like this:
# obj <- read_plink("data")
# this also works
# obj <- read_plink("data.bed")
#
# you get a list with these three items:
# genotypes
# obj$X
```

```
# locus annotations
# obj$bim
# individual annotations
# obj$fam

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# first get path to BED file
file <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)

# read genotypes and annotation tables
plink_data <- read_plink(file)
# genotypes
plink_data$X
# locus annotations
plink_data$bim
# individual annotations
plink_data$fam

# the same works without .bed extension
file <- sub('\\.bed$', '', file) # remove extension
# it works!
plink_data <- read_plink(file)
```

`read_snp` *Read Eigenstrat *.snp files*

Description

This function reads a standard Eigenstrat *.snp file into a tibble. It uses `readr::read_table()` to do it efficiently.

Usage

```
read_snp(file, verbose = TRUE)
```

Arguments

`file`	Input file (whatever is accepted by `readr::read_table()`). If file as given does not exist and is missing the expected *.snp extension, the function adds the .snp extension and uses that path if that file exists. Additionally, the .gz extension is added automatically if the file (after *.snp extension is added as needed) is still not found and did not already contain the .gz extension and adding it points to an existing file.
`verbose`	If `TRUE` (default), function reports the path of the file being loaded (after autocompleting the extensions).

Value

A tibble with columns: `id`, `chr`, `posg`, `pos`, `ref`, `alt`

See Also

Eigenstrat SNP format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

Examples

```
# to read "data.snp", run like this:
# snp <- read_snp("data")
# this also works
# snp <- read_snp("data.snp")

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# read an existing Eigenstrat *.snp file
file <- system.file("extdata", 'sample.snp', package = "genio", mustWork = TRUE)
snp <- read_snp(file)
snp

# can specify without extension
file <- sub('\\.snp$', '', file) # remove extension from this path on purpose
file # verify .snp is missing
snp <- read_snp(file) # load it anyway!
snp
```

`require_files_grm` *Require that GCTA binary GRM files are present*

Description

This function checks that each of the GCTA binary GRM files (`grm.bin`, `grm.N.bin`, and `grm.id` extensions) are present, given the shared base file path, stopping with an informative message if any of the files is missing. This function aids troubleshooting, as various downstream external software report missing files differently and sometimes using confusing or obscure messages.

Usage

```
require_files_grm(file)
```

Arguments

`file`	The shared file path (excluding extensions: `grm.bin`, `grm.N.bin`, or `grm.id`).

Value

Nothing

Examples

```
# to require all of "data.grm.bin", "data.grm.N.bin", and "data.grm.id", run like this:
# (stops if any of the three files is missing)
# require_files_grm("data")

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# check that the samples we want exist
# start with bed file
file <- system.file("extdata", 'sample.grm.bin', package = "genio", mustWork = TRUE)
# remove extension
file <- sub('\\.grm\\.bin$', '', file)
# since all sample.grm.{bin,N.bin,id} files exist, this will not stop with error messages:
require_files_grm(file)
```

`require_files_phen` *Require that PHEN file is present*

Description

This function checks that the PHEN file is present, given the base file path, stopping with an informative message if the file is missing. This function aids troubleshooting, as various downstream external software report missing files differently and sometimes using confusing or obscure messages.

Usage

```
require_files_phen(file)
```

Arguments

`file`	The base file path (excluding phen extensions).

Value

Nothing

Examples

```
# to require "data.phen", run like this:
# (stops if file is missing)
# require_files_phen("data")

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# check that the samples we want exist
```

```
# get path to an existing phen file
file <- system.file("extdata", 'sample.phen', package = "genio", mustWork = TRUE)
# remove extension
file <- sub('\\.phen$', '', file)
# since sample.phen file exist, this will not stop with error messages:
require_files_phen(file)
```

`require_files_plink` *Require that Plink binary files are present*

Description

This function checks that each of the Plink binary files (BED/BIM/FAM extensions) are present, given the shared base file path, stopping with an informative message if any of the files is missing. This function aids troubleshooting, as various downstream external software report missing files differently and sometimes using confusing or obscure messages.

Usage

```
require_files_plink(file)
```

Arguments

`file`	The shared file path (excluding extensions `bed`, `bim`, `fam`).

Value

Nothing

Examples

```
# to require all of "data.bed", "data.bim", and "data.fam", run like this:
# (stops if any of the three files is missing)
# require_files_plink("data")

# The following example is more awkward
# because package sample data has to be specified in this weird way:

# check that the samples we want exist
# start with bed file
file <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)
# remove extension
file <- sub('\\.bed$', '', file)
# since all sample.{bed,bim,fam} files exist, this will not stop with error messages:
require_files_plink(file)
```

sex_to_char *Convert integer sex codes to character codes*

Description

This function accepts the integer sex codes accepted by Plink and turns them into the character codes accepted by Eigenstrat. Only upper-case characters are returned. Cases outside the table below are mapped to U (unknown) with a warning. The correspondence is:

- 0: U (unknown)
- 1: M (male)
- 2: F (female)

Usage

```
sex_to_char(sex)
```

Arguments

sex	Integer vector of sex codes

Value

The converted character vector of sex codes

See Also

sex_to_int()

Eigenstrat IND format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

Examples

```
# verify the mapping above
sex_int <- 0:2
sex_char <- c('U', 'M', 'F') # expected values
stopifnot(
  all(
    sex_to_char( sex_int ) == sex_char
  )
)
```

sex_to_int *Convert character sex codes to integer codes*

Description

This function accepts the character sex codes accepted by Eigenstrat and turns them into the integer codes accepted by Plink. Matching is case insensitive. Cases outside the table below are mapped to `0` (unknown) with a warning. The correspondence is:

- `U`: `0` (unknown)
- `M`: `1` (male)
- `F`: `2` (female)

Usage

```
sex_to_int(sex)
```

Arguments

`sex`	Character vector of sex codes

Value

The converted numeric vector of sex codes

See Also

`sex_to_char()`

Eigenstrat IND format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

Examples

```
# verify the mapping above
sex_char <- c('U', 'm', 'f') # mixed case works!
sex_int <- 0:2 # expected values
stopifnot(
  all(
    sex_to_int( sex_char ) == sex_int
  )
)
```

`sim_and_write_plink` *Simulate and write genotypes to plink format on the fly*

Description

To save memory, simulate small chunks of variants at the time and write them to file as you go. This is a wrapper around `write_plink()` and `readr::write_lines()` (for ancestral allele frequencies, optional) with `append = TRUE` that simplifies looping somewhat. The function always appends to the output, so it can be called several times if convenient, for example to simulate separate chromosomes.

Usage

```
sim_and_write_plink(
  sim_chunk,
  m_loci,
  fam,
  file,
  file_p_anc = NA,
  n_data_cut = 10^6
)
```

Arguments

`sim_chunk`	A function that generates a small number of loci at the time, to be called iteratively until the whole genome is complete. It should accept a single parameter, the number of loci to simulate at one time, and returns a list with these named elements: • `X`: the simulated genotype matrix, with the desired number of loci and the same individuals in every call. Required. • `bim`: the simulated variant table for the loci that were just simulated. Required. • `p_anc`: the vector of ancestral allele frequencies (required for simulating traits with correctly specified heritabilities). Optional.
`m_loci`	Total number of loci to simulate.
`fam`	Sample table of simulation to write.
`file`	Output file path, without extensions (each of .bed, .bim, .fam extensions will be added automatically as needed).
`file_p_anc`	Complete file path (with extensions) of vector of ancestral allele frequencies, if `sim_chunk` generates them (optional). This file is created with `readr::write_lines()`, so it is a plain text file with each line being the ancestral allele frequency of each locus in order, and it may be compressed if this file has a .gz extension.
`n_data_cut`	Number of cells (individuals times loci) to aim to simulate at the time. Actual number may be smaller to ensure that the number of loci is an integer, except if the number of individuals is greater than `n_data_cut` then a single locus will be simulated at the time (and the number of cells will be greater than `n_data_cut`).

See Also

write_plink()

Examples

```
# some global constants that will be accessed by simulator function
n <- 10
# and a global variable updated as we go
m_last <- 0

# define a trivial but complete genotype simulator function
my_sim_chunk <- function( m_chunk ) {
    # construct ancestral allele frequencies
    p_anc <- runif( m_chunk )
    # simulate genotypes from HWE
    X <- matrix( rbinom( m_chunk * n, 2, p_anc ), m_chunk, n )
    # construct a trivial BIM table
    bim <- make_bim( n = m_chunk )
    # but make sure count continues across chunks without repeats
    # (so IDs and positions don't clash!)
    bim$id <- m_last + ( 1 : m_chunk )
    # update global value (use <<-) for next round
    m_last <<- m_last + m_chunk
    # return all of these elements in a named list!
    return( list( X = X, bim = bim, p_anc = p_anc ) )
}

# the fam table is created fully now
fam <- make_fam( n = n )
# set other parameters
m_loci <- 100

# this is only necessary for example files to be in a *temporary* location
# (don't use `tempfile` in real cases)
# plink files path without extension
file <- tempfile('test')
# p_anc file should have extension
filep <- tempfile('test-p-anc.txt.gz')

# simulate and write as we go!
sim_and_write_plink( my_sim_chunk, m_loci, fam, file, filep )

# clean up: delete sample outputs
delete_files_plink( file )
file.remove( filep )
```

---

symlink *Create a symbolic link to a file, adjusting paths automatically if needed*

Description

This function creates a symbolic (soft) link to a file, in a solution that works for all major operating systems, so a file can have two names without actually duplicating data. Although the two paths can be specified directly, this function automatically handles a conversion for a common but troublesome case when the link is not in the current directory, in which case the file must be relative to the parent directory of the link, although it is more natural to specify the file relative to the current directory.

Usage

```
symlink(file, link, adjust_path = TRUE, verbose = TRUE)
```

Arguments

file	The file that will be linked. This function does not require this file to exist, but the link will be broken in that case.
link	The path to the link to the file. If this points to an existing file, or an existing link, it will be overwritten.
adjust_path	If TRUE (default), file is automatically adjusted in the special case in which it is a relative path (assumed to be relative to current directory) but link is not in the current directory, in which case file is adjusted to be relative to the parent directory of link. If file is an absolute path, it is never edited, and likewise no editing is needed if link is in the current directory. Set to FALSE to avoid editing in all cases.
verbose	If TRUE (default), function reports the link and the final file it points to.

Examples

```
# in this example, for the existing file, use this file provided by the package.
# Note that it is an absolute path, so it will not be edited.
file <- system.file("extdata", 'sample.bed', package = "genio", mustWork = TRUE)
# this is the path to the link
link <- tempfile('delete-me-example', fileext = '.bed')

# create the symbolic link!
symlink( file, link )

# delete example link when done
file.remove( link )
```

---

tidy_kinship *Create a tidy version of a kinship matrix*

Description

A square symmetric kinship matrix is transformed into a tibble, with a row per unique element in the kinship matrix, and three columns: ID of row, ID of column, and the kinship value.

Usage

```
tidy_kinship(kinship, sort = TRUE)
```

Arguments

kinship	The n-by-n symmetric kinship matrix
sort	If TRUE (default), rows are sorted ascending by kinship value. Otherwise, order is moving along the upper triangle row-by-row

Value

A tibble with `n * ( n + 1 ) / 2` rows (the upper triangle, including the diagonal), and 3 columns with names: `id1`, `id2`, `kinship`.

Examples

```
# create a symmetric matrix
kinship <- matrix(
    c(
        0.5, 0.1, 0.0,
        0.1, 0.5, 0.2,
        0.0, 0.2, 0.6
    ),
    nrow = 3
)
# add names (best for tidy version)
colnames(kinship) <- paste0('pop', 1:3)
rownames(kinship) <- paste0('pop', 1:3)
# this returns tidy version
kinship_tidy <- tidy_kinship( kinship )
# test colnames
stopifnot( colnames( kinship_tidy ) == c('id1', 'id2', 'kinship') )
# test row number
stopifnot( nrow( kinship_tidy ) == 6 )
# inspect it
kinship_tidy
```

---

write_bed *Write a genotype matrix into Plink BED format*

Description

This function accepts a standard R matrix containing genotypes (values in `c( 0, 1, 2, NA )`) and writes it into a Plink-formatted BED (binary) file. Each genotype per locus (`m` loci) and individual (`n` total) counts the number of alternative alleles or `NA` for missing data. No *.fam or *.bim files are created by this basic function.

Usage

```
write_bed(file, X, verbose = TRUE, append = FALSE)
```

Arguments

file	Output file path. .bed extension may be omitted (will be added automatically if it is missing).
X	The m-by-n genotype matrix. Row and column names, if present, are ignored.
verbose	If TRUE (default), function reports the path of the file being written (after autocompleting the extension).
append	If TRUE, appends variants onto the file. (Default is FALSE).

Details

Genotypes with values outside of [0, 2] cause an error, in which case the partial output is deleted. However, beware that decimals get truncated internally, so values that truncate to 0, 1, or 2 will not raise errors. The BED format does not accept fractional dosages, so such data will not be written as expected.

Value

Nothing

See Also

write_plink() for writing a set of BED/BIM/FAM files.

Plink BED format reference: https://www.cog-genomics.org/plink/1.9/formats#bed

Examples

```
# to write an existing matrix `X` into file "data.bed", run like this:
# write_bed("data", X)
# this also works
# write_bed("data.bed", X)

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

file_out <- tempfile('delete-me-example', fileext = '.bed') # will also work without extension
# create 10 random genotypes
X <- rbinom(10, 2, 0.5)
# replace 3 random genotypes with missing values
X[sample(10, 3)] <- NA
# turn into 5x2 matrix
X <- matrix(X, nrow = 5, ncol = 2)
# write this data to file in BED format
# (only *.bed gets created, no *.fam or *.bim in this call)
write_bed(file_out, X)
# delete output when done
```

```
file.remove(file_out)
```

`write_bim` *Write Plink *.bim files*

Description

This function writes a tibble with the right columns into a standard Plink *.bim file. It uses `readr::write_tsv()` to do it efficiently.

Usage

```
write_bim(file, tib, verbose = TRUE, append = FALSE)
```

Arguments

`file`	Output file (whatever is accepted by `readr::write_tsv()`). If file is missing the expected *.bim extension, the function adds it.
`tib`	The tibble or data.frame to write. It must contain these columns: `chr`, `id`, `posg`, `pos`, `alt`, `ref`. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format.
`verbose`	If `TRUE` (default), function reports the path of the file being written (after autocompleting the extension).
`append`	If `TRUE`, appends rows onto the file. (Default is `FALSE`).

Value

The output `tib` invisibly (what `readr::write_tsv()` returns).

See Also

`write_plink()` for writing a set of BED/BIM/FAM files.

Plink BIM format references: https://www.cog-genomics.org/plink/1.9/formats#bim https://www.cog-genomics.org/plink/2.0/formats#bim

Examples

```
# to write an existing table `bim` into file "data.bim", run like this:
# write_bim("data", bim)
# this also works
# write_bim("data.bim", bim)

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location
```

```
# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    chr = 1:3,
    id = 1:3,
    posg = 0,
    pos = 1:3,
    alt = 'B',
    ref = 'A'
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.bim') # will also work without extension
# write the table out in *.bim format (no header, columns in right order)
write_bim(file_out, tib)

# example cleanup
file.remove(file_out)
```

`write_eigenvec` *Write eigenvectors table into a Plink-format file*

Description

This function writes eigenvectors in Plink 1 (same as GCTA) format (table with no header, with first two columns being `fam` and `id`), which is a subset of Plink 2 format (which optionally allows column names and does not require fam column). Main expected case is `eigenvec` passed as a numeric matrix and `fam` provided to complete first two missing columns. However, input `eigenvec` may also be a data.frame already containing the `fam` and `id` columns, and other reasonable intermediate cases are also handled. If both `eigenvec` and `fam` are provided and contain overlapping columns, those in `eigenvec` get overwritten with a warning.

Usage

```
write_eigenvec(
  file,
  eigenvec,
  fam = NULL,
  ext = "eigenvec",
  plink2 = FALSE,
  verbose = TRUE
)
```

Arguments

`file`	The output file name (possibly without extension)
`eigenvec`	A matrix or tibble containing the eigenvectors to include in the file. Column names other than `fam` and `id` can be anything and are all treated as eigenvectors (not written to file).

fam	An optional fam table, which is used to add the fam and id columns to eigenvec (which overwrite columns of the same name in eigenvec if present, after a warning is produced). Individuals in fam and eigenvec are assumed to be the same and in the same order.
ext	Output file extension. Since the general "covariates" file format in GCTA and Plink are the same as this, this function may be used to write more general covariates files if desired, in which case users may wish to change this extension for clarity.
plink2	If TRUE, prints a header in the style of plink2 (starts with hash, fam -> FID, id -> IID, and the default PCs are named PC1, PC2, etc. Returned data.frame will also have these names.
verbose	If TRUE (default), function reports the path of the file being written (after autocompleting the extension).

Value

Invisibly, the final eigenvec data.frame or tibble written to file, starting with columns fam and id (merged from the fam input, if it was passed) followed by the rest of columns in the input eigenvec. Column names are instead #FID, IID, etc if plink2 = TRUE.

See Also

read_eigenvec() for reading an eigenvec file.

Plink 1 eigenvec format reference: https://www.cog-genomics.org/plink/1.9/formats#eigenvec

Plink 2 eigenvec format reference: https://www.cog-genomics.org/plink/2.0/formats#eigenvec

GCTA eigenvec format reference: https://cnsgenomics.com/software/gcta/#PCA

Examples

```
# to write an existing matrix `eigenvec` and optional `fam` tibble into file "data.eigenvec",
# run like this:
# write_eigenvec("data", eigenvec, fam = fam)
# this also works
# write_eigenvec("data.eigenvec", eigenvec, fam = fam)

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

# create dummy eigenvectors matrix, in this case from a small identity matrix
# number of individuals
n <- 10
eigenvec <- eigen( diag( n ) )$vectors
# subset columns to use top 3 eigenvectors only
eigenvec <- eigenvec[ , 1:3 ]
# dummy fam data
library(tibble)
fam <- tibble( fam = 1:n, id = 1:n )

# write this data to .eigenvec file
```

```
# output path without extension
file <- tempfile('delete-me-example')
eigenvec_final <- write_eigenvec( file, eigenvec, fam = fam )
# inspect the tibble that was written to file (returned invisibly)
eigenvec_final

# remove temporary file (add extension before deletion)
file.remove( paste0( file, '.eigenvec' ) )
```

`write_fam` *Write Plink *.fam files*

Description

This function writes a tibble with the right columns into a standard Plink *.fam file. It uses `readr::write_tsv()` to do it efficiently.

Usage

```
write_fam(file, tib, verbose = TRUE)
```

Arguments

`file`	Output file (whatever is accepted by `readr::write_tsv()`). If file is missing the expected *.fam extension, the function adds it.
`tib`	The tibble or data.frame to write. It must contain these columns: `fam`, `id`, `pat`, `mat`, `sex`, `pheno`. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format.
`verbose`	If `TRUE` (default), function reports the path of the file being written (after autocompleting the extension).

Value

The output `tib` invisibly (what `readr::write_tsv()` returns).

See Also

`write_plink()` for writing a set of BED/BIM/FAM files.

Plink FAM format reference: https://www.cog-genomics.org/plink/1.9/formats#fam

Examples

```
# to write an existing table `fam` into file "data.fam", run like this:
# write_fam("data", fam)
# this also works
# write_fam("data.fam", fam)

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    fam = 1:3,
    id = 1:3,
    pat = 0,
    mat = 0,
    sex = 1,
    pheno = 1
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.fam') # will also work without extension
# write the table out in *.fam format (no header, columns in right order)
write_fam(file_out, tib)
# delete output when done
file.remove(file_out)
```

`write_grm` *Write GCTA GRM and related plink2 binary files*

Description

This function writes a GCTA Genetic Relatedness Matrix (GRM, i.e. kinship) set of files in their binary format, given a kinship matrix and, if available, the corresponding matrix of pair sample sizes (non-trivial under missingness) and individuals table. Setting some options allows writing plink2 binary kinship formats such as "king" (follow examples in `read_grm()`).

Usage

```
write_grm(
  name,
  kinship,
  M = NULL,
  fam = NULL,
  verbose = TRUE,
  ext = "grm",
  shape = c("triangle", "strict_triangle", "square"),
  size_bytes = 4
)
```

Arguments

name	The base name of the output files. Files with that base, plus shared extension (default "grm", see ext below), plus extensions .bin, .N.bin, and .id may be created depending on the data provided.
kinship	The symmetric n-times-n kinship matrix to write into file with extension .<ext>.bin.
M	The optional symmetric n-times-n matrix of pair sample sizes to write into file with extension .<ext>.N.bin.
fam	The optional data.frame or tibble with individual annotations (columns with names fam and id, subset of columns of Plink FAM) to write into file with extension .<ext>.id. If fam is NULL but kinship has non-NULL column or row names, these are used as the second (id) value in the output table (the first (fam) column is set to the missing value in this case).
verbose	If TRUE (default), function reports the path of the files being written.
ext	Shared extension for all three outputs (see name above; default "grm"). Another useful value is "king", to match the KING-robust format produced by plink2. If NA, no extension is added. If given ext is also present at the end of name, then it is not added again.
shape	The shape of the information to write (may be abbreviated). Default "triangle" assumes there are n*(n+1)/2 values to write corresponding to the upper triangle including the diagonal (required for GCTA GRM). "strict_triangle" assumes there are n*(n-1)/2 values to write corresponding to the upper triangle *excluding* the diagonal (best for plink2 KING-robust). Lastly, "square" assumes there are n*n values to write corresponding to the entire square matrix, ignoring symmetry.
size_bytes	The number of bytes per number encoded. Default 4 corresponds to GCTA GRM and plink2 "bin4", whereas plink2 "bin" requires a value of 8.

See Also

read_grm()

Examples

```
# to write existing data `kinship`, `M`, and `fam` into files "data.grm.bin" etc, run like this:
# write_grm("data", kinship, M = M, fam = fam )

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

# create dummy data to write
# kinship for 3 individuals
kinship <- matrix(
    c(
        0.6, 0.2, 0.0,
        0.2, 0.5, 0.1,
        0.0, 0.1, 0.5
    ),
```

```
    nrow = 3
)
# pair sample sizes matrix
M <- matrix(
    c(
        10, 9, 8,
         9, 9, 7,
         8, 7, 8
    ),
    nrow = 3
)
# individual annotations table
library(tibble)
fam <- tibble(
    fam = 1:3,
    id = 1:3
)
# dummy files to write and delete
name <- tempfile('delete-me-example') # no extension
# write the data now!
write_grm( name, kinship, M = M, fam = fam )
# delete outputs when done
delete_files_grm( name )
```

`write_ind` *Write Eigenstrat *.ind files*

Description

This function writes a tibble with the right columns into a standard Eigenstrat *.ind file. It uses `readr::write_tsv()` to do it efficiently.

Usage

```
write_ind(file, tib, verbose = TRUE)
```

Arguments

`file`	Output file (whatever is accepted by `readr::write_tsv()`). If file is missing the expected *.ind extension, the function adds it.
`tib`	The tibble or data.frame to write. It must contain these columns: `id`, `sex`, `label`. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format.
`verbose`	If `TRUE` (default), function reports the path of the file being written (after autocompleting the extension).

Value

The output `tib` invisibly (what `readr::write_tsv()` returns).

See Also

Eigenstrat IND format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

Examples

```
# to write an existing table `ind` into file "data.ind", run like this:
# write_ind("data", ind)
# this also works
# write_ind("data.ind", ind)

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    id = 1:3,
    sex = 1,
    label = 1
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.ind') # will also work without extension
# write the table out in *.ind format (no header, columns in right order)
write_ind(file_out, tib)
# delete output when done
file.remove(file_out)
```

---

write_matrix *Write a matrix to a file without row or column names*

Description

The inverse function of read_matrix(), this writes what is intended to be a numeric matrix to a tab-delimited file without row or column names present. It uses readr::write_tsv() to do it efficiently. Intended for outputs such as those of admixture inference approaches.

Usage

```
write_matrix(file, x, ext = "txt", verbose = TRUE, append = FALSE)
```

Arguments

file	Output file (whatever is accepted by readr::write_tsv()). If file is missing the expected extension (see below), the function adds it.
x	The matrix to write. Unlike read_matrix(), this is not in fact required to be a matrix or be strictly numeric; anything that coerces to tibble or data.frame is acceptable.

ext	The desired file extension. If NA, no extension is added. Works if file already contains desired extension.
verbose	If TRUE (default), function reports the path of the file being written (after autocompleting the extension).
append	If TRUE, appends rows onto the file. (Default is FALSE).

Value

The output x, coerced into data.frame, invisibly (what readr::write_tsv() returns).

See Also

read_matrix(), the inverse function.

Examples

```
# to write an existing matrix `x` into file "data.txt", run like this:
# write_matrix( "data", x )
# this also works
# write_matrix( "data.txt", x )

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

# create a dummy matrix with the right columns
x <- rbind( 1:3, (0:2)/10, -1:1 )
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.txt') # will also work without extension
# write the matrix without header
write_matrix( file_out, x )
# delete output when done
file.remove( file_out )
```

---

write_phen *Write *.phen files*

Description

This function writes a tibble with the right columns into a standard *.phen file. It uses readr::write_tsv() to do it efficiently. GCTA and EMMAX use this format.

Usage

```
write_phen(file, tib, verbose = TRUE)
```

Arguments

file	Output file (whatever is accepted by readr::write_tsv()). If file is missing the expected *.phen extension, the function adds it.
tib	The tibble or data.frame to write. It must contain these columns: fam, id, pheno. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format.
verbose	If TRUE (default), function reports the path of the file being written (after autocompleting the extension).

Value

The output tib invisibly (what readr::write_tsv() returns).

See Also

GCTA PHEN format reference: https://cnsgenomics.com/software/gcta/#GREMLanalysis

Examples

```
# to write an existing table `phen` into file "data.phen", run like this:
# write_phen("data", phen)
# this also works
# write_phen("data.phen", phen)

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    fam = 1:3,
    id = 1:3,
    pheno = 1
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.phen') # will also work without extension
# write the table out in *.phen format (no header, columns in right order)
write_phen(file_out, tib)
# delete output when done
file.remove(file_out)
```

---

write_plink *Write genotype and sample data into a Plink BED/BIM/FAM file set.*

Description

This function writes a genotype matrix (X) and its associated locus (bim) and individual (fam) data tables into three Plink files in BED, BIM, and FAM formats, respectively. This function is a wrapper around the more basic functions write_bed(), write_bim(), write_fam(), but additionally tests that the data dimensions agree (or stops with an error). Also checks that the genotype row and column names agree with the bim and fam tables if they are all present. In addition, if bim = NULL or fam = NULL, these are auto-generated using make_bim() and make_fam(), which is useful behavior for simulated data. Lastly, the phenotype can be provided as a separate argument and incorporated automatically if fam = NULL (a common scenario for simulated genotypes and traits). Below suppose there are m loci and n individuals.

Usage

```
write_plink(
  file,
  X,
  bim = NULL,
  fam = NULL,
  pheno = NULL,
  verbose = TRUE,
  append = FALSE,
  write_phen = FALSE
)
```

Arguments

file	Output file path, without extensions (each of .bed, .bim, .fam extensions will be added automatically as needed).
X	The m-by-n genotype matrix.
bim	The tibble or data.frame containing locus information. It must contain m rows and these columns: chr, id, posg, pos, ref, alt. If NULL (default), it will be quietly auto-generated.
fam	The tibble or data.frame containing individual information. It must contain n rows and these columns: fam, id, pat, mat, sex, pheno. If NULL (default), it will be quietly auto-generated.
pheno	The phenotype to write into the FAM file assuming fam = NULL. This must be a length-n vector. This will be ignored (with a warning) if fam is provided.
verbose	If TRUE (default) function reports the paths of the files being written (after autocompleting the extensions).
append	If TRUE, appends loci onto the BED and BIM files (default FALSE). In this mode, all individuals must be present in each write (only loci are appended); the FAM file is not overwritten if present, but is required at every write for internal validations. If the FAM file already exists, it is not checked to agree with the FAM table provided. PHEN file is always unchanged and ignored if append = TRUE.
write_phen	If TRUE and append = FALSE, writes a .phen file too from the fam data provided or auto-generated (using write_phen()). Default FALSE.

Value

Invisibly, a named list with items in this order: `X` (genotype matrix), `bim` (tibble), `fam` (tibble). This is most useful when either BIM or FAM tables were auto-generated.

See Also

`write_bed()`, `write_bim()`, `write_fam()`, `make_bim()`, `make_fam()`.

Plink BED/BIM/FAM format reference: https://www.cog-genomics.org/plink/1.9/formats

Examples

```
# to write existing data `X`, `bim`, `fam` into files "data.bed", "data.bim", and "data.fam",
# run like this:
# write_plink("data", X, bim = bim, fam = fam)

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

# here is an example for a simulation

# create 10 random genotypes
X <- rbinom(10, 2, 0.5)
# replace 3 random genotypes with missing values
X[sample(10, 3)] <- NA
# turn into 5x2 matrix
X <- matrix(X, nrow = 5, ncol = 2)

# simulate a trait for two individuals
pheno <- rnorm(2)

# write this data to BED/BIM/FAM files
# output path without extension
file_out <- tempfile('delete-me-example')
# here all of the BIM and FAM columns except `pheno` are autogenerated
write_plink(file_out, X, pheno = pheno)

# delete all three outputs when done
delete_files_plink( file_out )
```

---

write_snp *Write Eigenstrat *.snp files*

Description

This function writes a tibble with the right columns into a standard Eigenstrat *.snp file. It uses `readr::write_tsv()` to do it efficiently.

Usage

```
write_snp(file, tib, verbose = TRUE)
```

Arguments

file	Output file (whatever is accepted by readr::write_tsv()). If file is missing the expected *.snp extension, the function adds it.
tib	The tibble or data.frame to write. It must contain these columns: id, chr, posg, pos, ref, alt. Throws an error if any of these columns are missing. Additional columns are ignored. Columns are automatically reordered in output as expected in format.
verbose	If TRUE (default), function reports the path of the file being written (after autocompleting the extension).

Value

The output tib invisibly (what readr::write_tsv() returns).

See Also

Eigenstrat SNP format reference: https://github.com/DReichLab/EIG/tree/master/CONVERTF

Examples

```
# to write an existing table `snp` into file "data.snp", run like this:
# write_snp("data", snp)
# this also works
# write_snp("data.snp", snp)

# The following example is more detailed but also more awkward
# because (only for these examples) the package must create the file in a *temporary* location

# create a dummy tibble with the right columns
library(tibble)
tib <- tibble(
    id = 1:3,
    chr = 1:3,
    posg = 0,
    pos = 1:3,
    ref = 'A',
    alt = 'B'
)
# a dummy file
file_out <- tempfile('delete-me-example', fileext = '.snp') # will also work without extension
# write the table out in *.snp format (no header, columns in right order)
write_snp(file_out, tib)
# delete output when done
file.remove(file_out)
```

Index