

Package ‘BGmisc’

January 11, 2026

Title An R Package for Extended Behavior Genetics Analysis

Version 1.5.2

Description Provides functions for behavior genetics analysis,
including variance component model identification [Hunter et al. (2021) <[doi:10.1007/s10519-021-10055-x](https://doi.org/10.1007/s10519-021-10055-x)>],
calculation of relatedness coefficients using path-tracing methods
[Wright (1922) <[doi:10.1086/279872](https://doi.org/10.1086/279872)>; McArdle & McDonald (1984) <[doi:10.1111/j.2044-8317.1984.tb00802.x](https://doi.org/10.1111/j.2044-8317.1984.tb00802.x)>],
inference of relatedness, pedigree conversion, and simulation of multi-generational family data
[Lyu et al. (2024) <[doi:10.1101/2024.12.19.629449](https://doi.org/10.1101/2024.12.19.629449)>]. For a full overview,
see [Garrison et al. (2024) <[doi:10.21105/joss.06203](https://doi.org/10.21105/joss.06203)>].

License GPL-3

URL <https://github.com/R-Computing-Lab/BGmisc/>,
<https://r-computing-lab.github.io/BGmisc/>

BugReports <https://github.com/R-Computing-Lab/BGmisc/issues>

Depends R (>= 3.5.0)

Imports data.table, igraph, Matrix, stats, stringr, methods

Suggests corplot, discord, dplyr, EasyMx, ggpedigree, ggplot2,
kinship2, knitr, OpenMx, rmarkdown, testthat (>= 3.0.0),
tidyverse, withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author S. Mason Garrison [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-4804-6003>>),
Michael D. Hunter [aut] (ORCID:

<https://orcid.org/0000-0002-3651-6709>),
 Xuanyu Lyu [aut] (ORCID: <https://orcid.org/0000-0002-2841-5529>),
 Rachel N. Good [ctb],
 Jonathan D. Trattner [aut] (ORCID:
<https://orcid.org/0000-0002-1097-7603>),
 S. Alexandra Burt [aut] (ORCID:
<https://orcid.org/0000-0001-5538-7431>)

Maintainer S. Mason Garrison <garrissm@wfu.edu>

Repository CRAN

Date/Publication 2026-01-11 06:12:42 UTC

Contents

.adjBeta	4
.adjDirect	5
.adjIndexed	6
.assignParentValue	7
.computeTranspose	8
.loadOrComputeIsChild	8
.postProcessGedcom.legacy	9
addParentRow	9
addPersonToPed	10
addRowlessParents	11
applyTagMappings	12
buildBetweenGenerations	12
buildWithinGenerations	14
calcAllGens	15
calcFamilySize	16
calcFamilySizeByGen	16
calculateCIs	17
calculateH	18
calculateRelatedness	19
checkIDs	21
checkIDUniqueness	21
checkParentIDs	22
checkParentSex	23
checkPedigreeNetwork	24
checkSex	25
checkWithinRowDuplicates	26
collapseNames	27
com2links	27
comp2vech	28
computeParentAdjacency	29
countPatternRows	31
createGenDataFrame	31
determineSex	32
dropLink	33

findBiggest	33
findOldest	34
fitComponentModel	35
hazard	36
identifyComponentModel	37
inbreeding	38
initializeRecord	39
insertEven	39
makeInbreeding	40
makeTwins	41
mapFAMS2parents	42
markPotentialChildren	42
parseNameLine	43
ped2add	44
ped2ce	45
ped2cn	46
ped2com	47
ped2fam	49
ped2gen	50
ped2graph	51
ped2maternal	52
ped2mit	53
ped2paternal	55
postProcessGedcom	56
potter	56
prepSummarizePedigrees	57
processEventLine	58
processParents	59
readGedcom	59
readWikifamilytree	61
recodeSex	62
repairIDs	64
repairParentIDs	64
repairSex	65
royal92	66
simulatePedigree	67
sliceFamilies	69
summarizeFamilies	70
summarizeMatrilines	72
summarizePatrilines	74
summarizePedigrees	75
traceTreePaths	78
validate_and_convert_matrix	78
vech	79

<code>.adjBeta</code>	<i>Construct Adjacency Matrix for Parent-Child Relationships Using Beta Method This function constructs an adjacency matrix for parent-child relationships using a method in beta testing. It identifies parent-child pairs based on the specified component of relatedness.</i>
-----------------------	--

Description

Construct Adjacency Matrix for Parent-Child Relationships Using Beta Method This function constructs an adjacency matrix for parent-child relationships using a method in beta testing. It identifies parent-child pairs based on the specified component of relatedness.

Usage

```
.adjBeta(
  ped,
  component,
  adjBeta_method = 5,
  parList = NULL,
  lastComputed = 0,
  lens = NULL,
  saveable = FALSE,
  resume = FALSE,
  save_path = NULL,
  verbose = FALSE,
  save_rate_parlist = NULL,
  update_rate = NULL,
  checkpoint_files = NULL,
  config,
  compress = config$compress,
  ...
)
```

Arguments

<code>ped</code>	a pedigree dataset. Needs ID, momID, and dadID columns
<code>component</code>	character. Which component of the pedigree to return. See Details.
<code>adjBeta_method</code>	numeric The method to use for computing the building the adjacency_method matrix when using the "beta" build
<code>parList</code>	a list of parent-child relationships
<code>lastComputed</code>	the last computed index
<code>lens</code>	a vector of the lengths of the parent-child relationships
<code>saveable</code>	logical. If TRUE, save the intermediate results to disk
<code>resume</code>	logical. If TRUE, resume from a checkpoint
<code>save_path</code>	character. The path to save the checkpoint files

<code>verbose</code>	logical. If TRUE, print progress through stages of algorithm
<code>save_rate_parlist</code>	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to <code>save_rate*1000</code>
<code>update_rate</code>	numeric. The rate at which to print progress
<code>checkpoint_files</code>	a list of checkpoint files
<code>config</code>	a configuration list that passes parameters to the function
<code>compress</code>	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.
<code>...</code>	additional arguments to be passed to ped2com

<code>.adjDirect</code>	<i>Construct Adjacency Matrix for Parent-Child Relationships Using Direct Method</i>
-------------------------	--

Description

This function constructs an adjacency matrix for parent-child relationships using a direct method. It identifies parent-child pairs based on the specified component of relatedness.

Usage

```
.adjDirect(
  ped,
  component,
  saveable,
  resume,
  save_path,
  verbose,
  lastComputed,
  checkpoint_files,
  update_rate,
  parList,
  lens,
  save_rate_parlist,
  config,
  compress = config$compress,
  ...
)
```

Arguments

<code>ped</code>	a pedigree dataset. Needs ID, momID, and dadID columns
<code>component</code>	character. Which component of the pedigree to return. See Details.

saveable	logical. If TRUE, save the intermediate results to disk
resume	logical. If TRUE, resume from a checkpoint
save_path	character. The path to save the checkpoint files
verbose	logical. If TRUE, print progress through stages of algorithm
lastComputed	the last computed index
checkpoint_files	a list of checkpoint files
update_rate	numeric. The rate at which to print progress
parList	a list of parent-child relationships
lens	a vector of the lengths of the parent-child relationships
save_rate_parlist	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to save_rate*1000
config	a configuration list that passes parameters to the function
compress	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.
...	additional arguments to be passed to ped2com

<i>.adjIndexed</i>	<i>Construct Adjacency Matrix for Parent-Child Relationships Using Indexed Method</i>
--------------------	---

Description

Construct Adjacency Matrix for Parent-Child Relationships Using Indexed Method

Usage

```
.adjIndexed(
  ped,
  component,
  saveable,
  resume,
  save_path,
  verbose,
  lastComputed,
  checkpoint_files,
  update_rate,
  parList,
  lens,
  save_rate_parlist,
  config,
  compress = config$compress
)
```

Arguments

<code>ped</code>	a pedigree dataset. Needs ID, momID, and dadID columns
<code>component</code>	character. Which component of the pedigree to return. See Details.
<code>saveable</code>	logical. If TRUE, save the intermediate results to disk
<code>resume</code>	logical. If TRUE, resume from a checkpoint
<code>save_path</code>	character. The path to save the checkpoint files
<code>verbose</code>	logical. If TRUE, print progress through stages of algorithm
<code>lastComputed</code>	the last computed index
<code>checkpoint_files</code>	a list of checkpoint files
<code>update_rate</code>	numeric. The rate at which to print progress
<code>parList</code>	a list of parent-child relationships
<code>lens</code>	a vector of the lengths of the parent-child relationships
<code>save_rate_parlist</code>	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to <code>save_rate*1000</code>
<code>config</code>	a configuration list that passes parameters to the function
<code>compress</code>	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.

<code>.assignParentValue</code>	<i>Assign parent values based on component type</i>
---------------------------------	---

Description

Assign parent values based on component type

Usage

```
.assignParentValue(component)
```

Arguments

<code>component</code>	character. Which component of the pedigree to return. See Details.
------------------------	--

<code>.computeTranspose</code>	<i>Compute the transpose multiplication for the relatedness matrix</i>
--------------------------------	--

Description

Compute the transpose multiplication for the relatedness matrix

Usage

```
.computeTranspose(r2, transpose_method = "tcrossprod", verbose = FALSE)
```

Arguments

<code>r2</code>	a relatedness matrix
<code>transpose_method</code>	character. The method to use for computing the transpose. Options are "tcrossprod", "crossprod", or "star"
<code>verbose</code>	logical. If TRUE, print progress through stages of algorithm

Details

The algorithms and methodologies used in this function are further discussed and exemplified in the vignette titled "examplePedigreeFunctions". For more advanced scenarios and detailed explanations, consult this vignette.

<code>.loadOrComputeIsChild</code>	<i>Load or compute the isChild matrix</i>
------------------------------------	---

Description

Load or compute the isChild matrix

Usage

```
.loadOrComputeIsChild(ped, checkpoint_files, config, compress = TRUE)
```

Arguments

<code>ped</code>	a pedigree dataset. Needs ID, momID, and dadID columns
<code>checkpoint_files</code>	A list of checkpoint file paths. @keywords internal
<code>config</code>	A list containing configuration parameters such as 'resume', 'verbose', and 'saveable'.
<code>compress</code>	a logical specifying whether saving to a named file is to use "gzip" compression, or one of "gzip", "bzip2", "xz" or "zstd" to indicate the type of compression to be used. Ignored if file is a connection.

.postProcessGedcom.legacy

Post-process GEDCOM Data Frame

Description

Post-process GEDCOM Data Frame

Usage

```
.postProcessGedcom.legacy(  
  df_temp,  
  remove_empty_cols = TRUE,  
  combine_cols = TRUE,  
  add_parents = TRUE,  
  skinny = TRUE,  
  verbose = FALSE  
)
```

Arguments

df_temp	A data frame containing information about individuals.
remove_empty_cols	A logical value indicating whether to remove columns with all missing values.
combine_cols	A logical value indicating whether to combine columns with duplicate values.
add_parents	A logical value indicating whether to add parents to the data frame.
skinny	A logical value indicating whether to return a skinny data frame.
verbose	A logical value indicating whether to print messages.

Value

A data frame with processed information.

addParentRow

Create a properly formatted parent row for the pedigree

Description

Create a properly formatted parent row for the pedigree

Usage

```
addParentRow(template_row, newID, sex, momID = NA, dadID = NA, famID = NA)
```

Arguments

template_row	A single row from ped, used as a template for column structure
newID	The new parent's ID
sex	The new parent's sex value (e.g., 0 for female, 1 for male, or "F"/"M")
momID	The new parent's mother ID (default is NA)
dadID	The new parent's father ID (default is NA)
famID	The new parent's family ID (default is NA)

Value

A single-row dataframe for the new parent

addPersonToPed	<i>addPersonToPed</i>
----------------	-----------------------

Description

A function to add a new person to an existing pedigree data.frame.

Usage

```
addPersonToPed(
  ped,
  name = NULL,
  sex = NULL,
  momID = NA,
  dadID = NA,
  twinID = NULL,
  personID = NULL,
  zygosity = NULL,
  notes = NULL,
  url = NULL,
  overwrite = FALSE
)
```

Arguments

ped	A data.frame representing the existing pedigree.
name	Optional. A character string representing the name of the new person. If not provided, the name will be set to NA.
sex	A value representing the sex of the new person.
momID	Optional. The ID of the mother of the new person. If not provided, it will be set to NA.
dadID	Optional. The ID of the father of the new person. If not provided, it will be set to NA.

twinID	Optional. The ID of the twin of the new person. If not provided, it will be set to NA.
personID	Optional. The ID of the new person. If not provided, it will be generated as the maximum existing personID + 1.
zygosity	Optional. A character string indicating the zygosity of the new person. If not provided, it will be set to NA.
notes	Optional. A character string for notes about the new person. If not provided, it will be set to NA.
url	Optional. A URL column for the new person. If not provided, it will be set to NA.
overwrite	Logical. If TRUE, the function will overwrite an existing person with the same personID. If FALSE, it will stop if a person with the same personID already exists.

Value

A data.frame with the new person added to the existing pedigree.

addRowlessParents	<i>Add addRowlessParents</i>
-------------------	------------------------------

Description

This function adds parents who appear in momID or dadID but are missing from ID

Usage

```
addRowlessParents(ped, verbose, validation_results)
```

Arguments

ped	A dataframe representing the pedigree data with columns 'ID', 'dadID', and 'momID'.
verbose	A logical flag indicating whether to print progress and validation messages to the console.
validation_results	validation results

applyTagMappings	<i>Apply Tag Mappings to a Line</i>
------------------	-------------------------------------

Description

Iterates over a list of tag mappings and, if a tag matches the line, updates the record. Stops after the first match.

Usage

```
applyTagMappings(line, record, pattern_rows, tag_mappings)
```

Arguments

line	A character string from the GEDCOM file.
record	A named list representing the individual's record.
pattern_rows	A list with GEDCOM tag counts.
tag_mappings	A list of lists. Each sublist should define: - tag: the GEDCOM tag, - field: the record field to update, - mode: either "replace" or "append", - extractor: (optional) a custom extraction function.

Value

A list with the updated record (record) and a logical flag (matched).

buildBetweenGenerations	<i>Process Generation Connections</i>
-------------------------	---------------------------------------

Description

This function processes connections between each two generations in a pedigree simulation. It marks individuals as parents, sons, or daughters based on their generational position and relationships. The function also handles the assignment of couple IDs, manages single and coupled individuals, and establishes parent-offspring links across generations.

Usage

```
buildBetweenGenerations(
  df_Fam,
  Ngen,
  sizeGens,
  verbose = FALSE,
  marR,
  sexR,
```

```

    kpc,
    rd_kpc,
    personID = "ID",
    momID = "momID",
    dadID = "dadID",
    code_male = "M",
    code_female = "F"
  )

```

Arguments

df_Fam	A data frame containing the simulated pedigree information up to the current generation. Must include columns for family ID, individual ID, generation number, spouse ID (spID), and sex. This data frame is updated in place to include flags for parental status (ifparent), son status (ifson), and daughter status (ifdau), as well as couple IDs.
Ngen	Number of generations. An integer ≥ 2 that determines how many generations the simulated pedigree will have. The first generation is always a fertilized couple. The last generation has no mated individuals.
sizeGens	A numeric vector containing the sizes of each generation within the pedigree.
verbose	logical If TRUE, message progress through stages of algorithm
marR	Mating rate. A numeric value ranging from 0 to 1 which determines the proportion of mated (fertilized) couples in the pedigree within each generation. For instance, $\text{marR} = 0.5$ suggests 50 percent of the offspring in a specific generation will be mated and have their offspring.
sexR	Sex ratio of offspring. A numeric value ranging from 0 to 1 that determines the proportion of males in all offspring in this pedigree. For instance, 0.4 means 40 percent of the offspring will be male.
kpc	Number of kids per couple. An integer ≥ 2 that determines how many kids each fertilized mated couple will have in the pedigree. Default value is 3. Returns an error when kpc equals 1.
rd_kpc	logical. If TRUE, the number of kids per mate will be randomly generated from a poisson distribution with mean kpc. If FALSE, the number of kids per mate will be fixed at kpc.
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
code_male	The value to use for males. Default is "M"
code_female	The value to use for females. Default is "F"

Details

The function iterates through each generation, starting from the second, to establish connections based on mating and parentage. For the first generation, it sets the parental status directly. For subsequent generations, it calculates the number of couples, the expected number of offspring,

and assigns offspring to parents. It handles gender-based assignments for sons and daughters, and deals with the nuances of single individuals and couple formation. The function relies on external functions ‘assignCoupleIds’ and ‘adjustKidsPerCouple’ to handle specific tasks related to couple ID assignment and offspring number adjustments, respectively.

Value

The function updates the ‘df_Fam’ data frame in place, adding or modifying columns related to parental and offspring status, as well as assigning unique couple IDs. It does not return a value explicitly.

buildWithinGenerations

Process Generations for Pedigree Simulation

Description

This function iterates through generations in a pedigree simulation, assigning IDs, creating data frames, determining sexes, and managing pairing within each generation.

Usage

```
buildWithinGenerations(
  sizeGens,
  marR,
  sexR,
  Ngen,
  verbose = FALSE,
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  code_male = "M",
  code_female = "F",
  fam_shift = 1L
)
```

Arguments

sizeGens	A numeric vector containing the sizes of each generation within the pedigree.
marR	Mating rate. A numeric value ranging from 0 to 1 which determines the proportion of mated (fertilized) couples in the pedigree within each generation. For instance, marR = 0.5 suggests 50 percent of the offspring in a specific generation will be mated and have their offspring.
sexR	Sex ratio of offspring. A numeric value ranging from 0 to 1 that determines the proportion of males in all offspring in this pedigree. For instance, 0.4 means 40 percent of the offspring will be male.

Ngen	Number of generations. An integer ≥ 2 that determines how many generations the simulated pedigree will have. The first generation is always a fertilized couple. The last generation has no mated individuals.
verbose	logical If TRUE, message progress through stages of algorithm
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
code_male	The value to use for males. Default is "M"
code_female	The value to use for females. Default is "F"
fam_shift	An integer to shift the person ID. Default is 1L.

Value

A data frame representing the simulated pedigree, including columns for family ID ('fam'),

calcAllGens	<i>calcAllGens A function to calculate the number of individuals in each generation. This is a supporting function for simulatePedigree.</i>
-------------	--

Description

calcAllGens A function to calculate the number of individuals in each generation. This is a supporting function for simulatePedigree.

Usage

```
calcAllGens(kpc, Ngen, marR)
```

```
allGens(kpc, Ngen, marR)
```

Arguments

kpc	Number of kids per couple (integer ≥ 2).
Ngen	Number of generations (integer ≥ 1).
marR	Mating rate (numeric value ranging from 0 to 1).

Value

Returns a vector containing the number of individuals in every generation.

calcFamilySize	<i>calcFamilySize</i> A function to calculate the total number of individuals in a pedigree given parameters. This is a supporting function for function simulatePedigree
----------------	---

Description

calcFamilySize A function to calculate the total number of individuals in a pedigree given parameters. This is a supporting function for function simulatePedigree

Usage

```
calcFamilySize(kpc, Ngen, marR)
```

```
famSizeCal(kpc, Ngen, marR)
```

Arguments

kpc	Number of kids per couple (integer ≥ 2).
Ngen	Number of generations (integer ≥ 1).
marR	Mating rate (numeric value ranging from 0 to 1).

Value

Returns a numeric value indicating the total pedigree size.

calcFamilySizeByGen	<i>calcFamilySizeByGen</i> An internal supporting function for simulatePedigree.
---------------------	--

Description

calcFamilySizeByGen An internal supporting function for simulatePedigree.

Usage

```
calcFamilySizeByGen(kpc, Ngen, marR)
```

```
sizeAllGens(kpc, Ngen, marR)
```

Arguments

kpc	Number of kids per couple (integer ≥ 2).
Ngen	Number of generations (integer ≥ 1).
marR	Mating rate (numeric value ranging from 0 to 1).

Value

Returns a vector including the number of individuals in every generation.

calculateCIs	<i>Confidence Intervals for Correlations with Optional Design-Effect Adjustment</i>
--------------	---

Description

Compute confidence intervals (CIs) for correlation coefficients using either Fisher's $r \rightarrow z$ approach (Raykov-style on the z scale) or a Wald CI on the r scale. Standard errors are first ****adjusted**** by a design-effect factor when available, and optionally for double entry. The adjusted standard errors are used for all calculations, including CIs, z-tests, and p-values.

Usage

```
calculateCIs(
  tbl,
  rho_var,
  se_var,
  doubleentered = FALSE,
  method = "raykov",
  adjust_base = 1,
  design_effect_m = NULL,
  design_effect_rho = NULL,
  design_effect_m_col = NULL,
  design_effect_rho_col = NULL,
  conf_level = 0.95
)
```

Arguments

tbl	A data frame or tibble containing the correlation coefficient and standard error variables.
rho_var	The name of the column in tbl that contains the correlation coefficients.
se_var	The name of the column in tbl that contains the standard errors.
doubleentered	Logical. If TRUE, the function assumes that the correlation coefficients are double-entered, which adjusts the standard errors accordingly. Default is FALSE.
method	Character; CI method selector. Supported values: <ul style="list-style-type: none"> • "raykov" — Fisher $r \rightarrow z$ CI (back-transformed). • "fisherz" — alias of "raykov". • "wald" — Wald CI on the r scale. • "doubleentered" — like "raykov" and, if doubleentered was not explicitly provided, it is set to TRUE (applies the $\sqrt{2}$ multiplier).

- "doubleenteredconserv" — like "wald" and, if doubleentered was not explicitly provided, it is set to TRUE.
- adjust_base A numeric value to adjust the standard errors. Default is 1.
- design_effect_m A numeric value for the design effect related to the mean. Default is NULL.
- design_effect_rho A numeric value for the design effect related to the correlation. Default is NULL.
- design_effect_m_col A character string specifying the column name for the design effect related to the mean. Default is NULL.
- design_effect_rho_col A character string specifying the column name for the design effect related to the correlation. Default is NULL.
- conf_level The confidence level for the intervals. Default is 0.95.

Value

A modified version of tbl with additional columns for the confidence intervals and related statistics. Everything uses adjusted standard errors, including confidence intervals, z-tests, and p-values.

Note

Double-entry handling and design effects are governed by doubleentered, design_effect_m/design_effect_rho (or their *_col variants), and adjust_base. The "doubleentered*" method values simply provide convenient aliases: they toggle doubleentered to TRUE only when the user hasn't explicitly set it, and map to "raykov" or "wald" as described.

Examples

```
tbl <- data.frame(rho = c(0.5, 0.7, 0.3), se = c(0.1, 0.2, 0.05))
calculateCIs(tbl, rho_var = "rho", se_var = "se", method = "raykov")
```

calculateH	<i>Falconer's Formula</i>
------------	---------------------------

Description

Use Falconer's formula to solve for H using the observed correlations for two groups of any two levels of relatednesses.

Usage

```
calculateH(r1, r2, obsR1, obsR2)
```

Arguments

r1	Relatedness coefficient of the first group.
r2	Relatedness coefficient of the second group.
obsR1	Observed correlation between members of the first group.
obsR2	Observed correlation between members of the second group.

Details

This generalization of Falconer's formula provides a method to calculate heritability by using the observed correlations for two groups of any two relatednesses. This function solves for H using the formula:

$$H^2 = \frac{obsR1 - obsR2}{r1 - r2}$$

where r1 and r2 are the relatedness coefficients for the first and second group, respectively, and obsR1 and obsR2 are the observed correlations.

Value

Heritability estimates ('heritability_estimates').

calculateRelatedness	<i>Calculate Relatedness Coefficient</i>
----------------------	--

Description

This function calculates the relatedness coefficient between two individuals based on their shared ancestry, as described by Wright (1922).

Usage

```
calculateRelatedness(
  generations = 2,
  path = NULL,
  full = TRUE,
  maternal = FALSE,
  empirical = FALSE,
  segregating = TRUE,
  total_a = 6800 * 1e+06,
  total_m = 16500,
  weight_a = 1,
  weight_m = 1,
  denom_m = FALSE,
  ...
)

related_coef(...)
```

Arguments

generations	Number of generations back of common ancestors the pair share.
path	Traditional method to count common ancestry, which is twice the number of generations removed from common ancestors. If not provided, it is calculated as $2 \times \text{generations}$.
full	Logical. Indicates if the kin share both parents at the common ancestor's generation. Default is TRUE.
maternal	Logical. Indicates if the maternal lineage should be considered in the calculation.
empirical	Logical. Adjusts the coefficient based on empirical data, using the total number of nucleotides and other parameters.
segregating	Logical. Adjusts for segregating genes.
total_a	Numeric. Represents the total size of the autosomal genome in terms of nucleotides, used in empirical adjustment. Default is 6800×1000000 .
total_m	Numeric. Represents the total size of the mitochondrial genome in terms of nucleotides, used in empirical adjustment. Default is 16500.
weight_a	Numeric. Represents the weight of phenotypic influence from additive genetic variance, used in empirical adjustment.
weight_m	Numeric. Represents the weight of phenotypic influence from mitochondrial effects, used in empirical adjustment.
denom_m	Logical. Indicates if 'total_m' and 'weight_m' should be included in the denominator of the empirical adjustment calculation.
...	Further named arguments that may be passed to another function.

Details

The relatedness coefficient between two people (b & c) is defined in relation to their common ancestors: $r_{bc} = \sum \left(\frac{1}{2}\right)^{n+n'+1} (1 + f_a)$

Value

Relatedness Coefficient ('coef'): A measure of the genetic relationship between two individuals.

Examples

```
## Not run:
# For full siblings, the relatedness coefficient is expected to be 0.5:
calculateRelatedness(generations = 1, full = TRUE)
# For half siblings, the relatedness coefficient is expected to be 0.25:
calculateRelatedness(generations = 1, full = FALSE)

## End(Not run)
```

checkIDs	<i>Validates and Optionally Repairs Unique IDs in a Pedigree Dataframe</i>
----------	--

Description

This function takes a pedigree object and performs two main tasks: 1. Checks for the uniqueness of individual IDs. 2. Optionally repairs non-unique IDs based on a specified logic.

Usage

```
checkIDs(ped, verbose = FALSE, repair = FALSE)
```

Arguments

ped	A dataframe representing the pedigree data with columns 'ID', 'dadID', and 'momID'.
verbose	A logical flag indicating whether to print progress and validation messages to the console.
repair	A logical flag indicating whether to attempt repairs on non-unique IDs.

Value

Depending on 'repair' value, either returns a list containing validation results or a repaired dataframe

Examples

```
## Not run:
ped <- data.frame(ID = c(1, 2, 2, 3), dadID = c(NA, 1, 1, 2), momID = c(NA, NA, 2, 2))
checkIDs(ped, verbose = TRUE, repair = FALSE)

## End(Not run)
```

checkIDUniqueness	<i>Check for duplicated individual IDs</i>
-------------------	--

Description

This function checks for duplicated individual IDs in a pedigree.

Usage

```
checkIDUniqueness(ped, verbose = FALSE)
```

Arguments

ped	A dataframe representing the pedigree data with columns 'ID', 'dadID', and 'momID'.
verbose	A logical flag indicating whether to print progress and validation messages to the console.

Value

A list containing the results of the check

checkParentIDs	<i>Validates and Optionally Repairs Parent IDs in a Pedigree Dataframe</i>
----------------	--

Description

This function takes a pedigree object and performs two main tasks: 1. Checks for the validity of parent IDs, specifically looking for instances where only one parent ID is missing. 2. Optionally repairs the missing parent IDs based on a specified logic.

Usage

```
checkParentIDs(
  ped,
  verbose = FALSE,
  repair = FALSE,
  repairsex = repair,
  addphantoms = repair,
  parentswithoutrow = repair,
  famID = "famID",
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  code_male = NULL,
  code_female = NULL
)
```

Arguments

ped	A dataframe representing the pedigree data with columns 'ID', 'dadID', and 'momID'.
verbose	A logical flag indicating whether to print progress and validation messages to the console.
repair	A logical flag indicating whether to attempt repairs on missing parent IDs.
repairsex	A logical flag indicating whether to attempt repairs on sex of the parents
addphantoms	A logical flag indicating whether to add phantom parents for missing parent IDs.

parentswithoutrow	A logical flag indicating whether to add parents without a row in the pedigree.
famID	Character. Column name for family IDs.
personID	Character. Column name for individual IDs.
momID	Character. Column name for maternal IDs.
dadID	Character. Column name for paternal IDs.
code_male	The code value used to represent male sex in the 'sex' column of ped.
code_female	The code value used to represent female sex in the 'sex' column of ped.

Value

Depending on the value of 'repair', either a list containing validation results or a repaired dataframe is returned.

Examples

```
## Not run:
ped <- data.frame(ID = 1:4, dadID = c(NA, 1, 1, 2), momID = c(NA, NA, 2, 2))
checkParentIDs(ped, verbose = TRUE, repair = FALSE)

## End(Not run)
```

checkParentSex	<i>Check Parental Role Sex Consistency</i>
----------------	--

Description

Validates sex coding consistency for a given parental role (momID or dadID).

Usage

```
checkParentSex(ped, parent_col, sex_col = "sex", verbose = FALSE)
```

Arguments

ped	Pedigree dataframe.
parent_col	The column name for parent IDs ("momID" or "dadID").
sex_col	The column name for sex coding. Default is "sex".
verbose	Logical, whether to print messages.

Value

A list containing role, unique sex codes, modal sex, inconsistent parents, and linked children.

`checkPedigreeNetwork` *Validate Pedigree Network Structure*

Description

Checks for structural issues in pedigree networks, including: - Individuals with more than two parents. - Presence of cyclic parent-child relationships.

Usage

```
checkPedigreeNetwork(  
  ped,  
  personID = "ID",  
  momID = "momID",  
  dadID = "dadID",  
  verbose = FALSE  
)
```

Arguments

<code>ped</code>	Dataframe representing the pedigree.
<code>personID</code>	Character. Column name for individual IDs.
<code>momID</code>	Character. Column name for maternal IDs.
<code>dadID</code>	Character. Column name for paternal IDs.
<code>verbose</code>	Logical. If TRUE, print informative messages.

Value

List containing detailed validation results.

Examples

```
## Not run:  
results <- checkPedigreeNetwork(ped,  
  personID = "ID",  
  momID = "momID", dadID = "dadID", verbose = TRUE  
)  
  
## End(Not run)
```


checkSex

*Validates and Optionally Repairs Sex Coding in a Pedigree Dataframe***Description**

This function checks and optionally modifies the coding of the biological 'sex' variable in a pedigree dataset. It serves two primary purposes: 1. Recodes the 'sex' variable based on specified codes for males and females, if provided. 2. Identifies and optionally repairs inconsistencies in sex coding that could break the algorithm for constructing genetic pedigrees.

Usage

```
checkSex(
  ped,
  code_male = NULL,
  code_female = NULL,
  code_unknown = NULL,
  verbose = FALSE,
  repair = FALSE,
  momID = "momID",
  dadID = "dadID"
)
```

Arguments

ped	A dataframe representing the pedigree data with a 'sex' column.
code_male	The current code used to represent males in the 'sex' column.
code_female	The current code used to represent females in the 'sex' column. If both are NULL, no recoding is performed.
code_unknown	The current code used to represent unknown or ambiguous sex in the 'sex' column. Can be NA to indicate that missing values should be treated as unknown. If NULL and both code_male and code_female are provided, values not matching either will be inferred as unknown.
verbose	A logical flag indicating whether to print progress and validation messages to the console.
repair	A logical flag indicating whether to attempt repairs on the sex coding.
momID	The column name for maternal IDs. Default is "momID".
dadID	The column name for paternal IDs. Default is "dadID".

Details

The validation process identifies: - The unique sex codes present in the dataset. - Whether individuals listed as fathers or mothers have inconsistent sex codes. - Instances where an individual's recorded sex does not align with their parental role.

If ‘repair = TRUE’, the function standardizes sex coding by: - Assigning individuals listed as fathers the most common male code in the dataset. - Assigning individuals listed as mothers the most common female code.

This function uses the terms ‘male’ and ‘female’ in a biological context, referring to chromosomal and other biologically-based characteristics necessary for constructing genetic pedigrees. The biological aspect of sex used in genetic analysis (genotype) is distinct from the broader, richer concept of gender identity (phenotype).

We recognize the importance of using language and methodologies that affirm and respect the full spectrum of gender identities. The developers of this package express unequivocal support for folx in the transgender and LGBTQ+ communities.

Value

Depending on the value of ‘repair’, either a list containing validation results or a repaired dataframe is returned.

Examples

```
## Not run:
ped <- data.frame(ID = c(1, 2, 3), sex = c("M", "F", "M"))
checkSex(ped, code_male = "M", verbose = TRUE, repair = FALSE)

## End(Not run)
```

checkWithinRowDuplicates

Check for within-row duplicates (self-parents, same mom/dad)

Description

This function checks for within-row duplicates in a pedigree.

Usage

```
checkWithinRowDuplicates(ped, verbose = FALSE)
```

Arguments

ped	A dataframe representing the pedigree data with columns ‘ID’, ‘dadID’, and ‘momID’.
verbose	A logical flag indicating whether to print progress and validation messages to the console.

Value

A list containing the results of the check

collapseNames	<i>collapse Names</i>
---------------	-----------------------

Description

This function combines the ‘name_given’ and ‘name_given_pieces’ columns in a data frame. If both columns have non-missing values that differ, a warning is issued and the original ‘name_given’ is retained. If one column is missing, the other is used. The same logic applies to the ‘name_surn’ and ‘name_surn_pieces’ columns.

Usage

```
collapseNames(verbose, df_temp)
```

Arguments

- verbose Logical. If TRUE, print progress messages.
- df_temp A data frame containing the columns to be combined.

Value

A data frame with the combined columns.

com2links	<i>Convert Sparse Relationship Matrices to Kinship Links</i>
-----------	--

Description

This function processes one or more sparse relationship components (additive, mitochondrial, and common nuclear) and converts them into kinship link pairs. The resulting related pairs are either returned as a data frame or written to disk in CSV format.

Usage

```
com2links(  
  rel_pairs_file = "dataRelatedPairs.csv",  
  ad_ped_matrix = NULL,  
  mit_ped_matrix = mt_ped_matrix,  
  mt_ped_matrix = NULL,  
  cn_ped_matrix = NULL,  
  write_buffer_size = 1000,  
  update_rate = 1000,  
  gc = TRUE,  
  writetodisk = TRUE,  
  verbose = FALSE,
```

```

    legacy = FALSE,
    outcome_name = "data",
    drop_upper_triangular = TRUE,
    include_all_links_1ped = FALSE,
    ...
  )

```

Arguments

rel_pairs_file File path to write related pairs to (CSV format).

ad_ped_matrix Matrix of additive genetic relatedness coefficients.

mit_ped_matrix Matrix of mitochondrial relatedness coefficients. Alias: **mt_ped_matrix**.

mt_ped_matrix Matrix of mitochondrial relatedness coefficients.

cn_ped_matrix Matrix of common nuclear relatedness coefficients.

write_buffer_size
Number of related pairs to write to disk at a time.

update_rate Numeric. Frequency (in iterations) at which progress messages are printed.

gc Logical. If TRUE, performs garbage collection via [gc](#) to free memory.

writetodisk Logical. If TRUE, writes the related pairs to disk; if FALSE, returns a data frame.

verbose Logical. If TRUE, prints progress messages.

legacy Logical. If TRUE, uses the legacy branch of the function.

outcome_name Character string representing the outcome name (used in file naming).

drop_upper_triangular
Logical. If TRUE, drops the upper triangular portion of the matrix.

include_all_links_1ped
Logical. If TRUE, includes all links in the output. (Default is true when only one ped is provided)

... Additional arguments to be passed to [com2links](#)

Value

A data frame of related pairs if **writetodisk** is FALSE; otherwise, writes the results to disk.

comp2vech	<i>comp2vech</i> Turn a variance component relatedness matrix into its half-vectorization
-----------	---

Description

comp2vech Turn a variance component relatedness matrix into its half-vectorization

Usage

```
comp2vech(x, include.zeros = FALSE)
```

Arguments

x Relatedness component matrix (can be a matrix, list, or object that inherits from 'Matrix').

include.zeros logical. Whether to include all-zero rows. Default is FALSE.

Details

This function is a wrapper around the vech function, extending it to allow for blockwise matrices and specific classes. It facilitates the conversion of a variance component relatedness matrix into a half-vectorized form.

Value

The half-vectorization of the relatedness component matrix.

Examples

```
comp2vech(list(matrix(c(1, .5, .5, 1), 2, 2), matrix(1, 2, 2)))
```

computeParentAdjacency

Compute Parent Adjacency Matrix with Multiple Approaches

Description

Compute Parent Adjacency Matrix with Multiple Approaches

Usage

```
computeParentAdjacency(
  ped,
  component,
  adjacency_method = "direct",
  saveable,
  resume,
  save_path,
  verbose = FALSE,
  lastComputed = 0,
  checkpoint_files,
  update_rate,
  parList,
  lens,
```

```

    save_rate_parlist,
    adjBeta_method = NULL,
    config,
    compress = config$compress,
    ...
)

```

Arguments

<code>ped</code>	a pedigree dataset. Needs ID, momID, and dadID columns
<code>component</code>	character. Which component of the pedigree to return. See Details.
<code>adjacency_method</code>	character. The method to use for computing the adjacency matrix. Options are "loop", "indexed", direct or beta
<code>saveable</code>	logical. If TRUE, save the intermediate results to disk
<code>resume</code>	logical. If TRUE, resume from a checkpoint
<code>save_path</code>	character. The path to save the checkpoint files
<code>verbose</code>	logical. If TRUE, print progress through stages of algorithm
<code>lastComputed</code>	the last computed index
<code>checkpoint_files</code>	a list of checkpoint files
<code>update_rate</code>	the rate at which to update the progress
<code>parList</code>	a list of parent-child relationships
<code>lens</code>	a vector of the lengths of the parent-child relationships
<code>save_rate_parlist</code>	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to <code>save_rate*1000</code>
<code>adjBeta_method</code>	numeric The method to use for computing the building the adjacency_method matrix when using the "beta" build
<code>config</code>	a configuration list that passes parameters to the function
<code>compress</code>	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.
<code>...</code>	additional arguments to be passed to ped2com

Details

The algorithms and methodologies used in this function are further discussed and exemplified in the vignette titled "examplePedigreeFunctions". For more advanced scenarios and detailed explanations, consult this vignette.

countPatternRows	<i>Count GEDCOM Pattern Rows</i>
------------------	----------------------------------

Description

Counts the number of lines in a file (passed as a data frame with column "X1") that match various GEDCOM patterns. Returns a list with counts for each pattern.

Usage

```
countPatternRows(file)
```

Arguments

file	A data frame with a column X1 containing GEDCOM lines.
------	--

Value

A list with counts of specific GEDCOM tag occurrences.

createGenDataFrame	<i>Create Data Frame for Generation</i>
--------------------	---

Description

This function creates a data frame for a specific generation within the simulated pedigree. It initializes the data frame with default values for family ID, individual ID, generation number, paternal ID, maternal ID, spouse ID, and sex. All individuals are initially set with NA for paternal, maternal, spouse IDs, and sex, awaiting further assignment.

Usage

```
createGenDataFrame(sizeGens, genIndex, idGen)
```

Arguments

sizeGens	A numeric vector containing the sizes of each generation within the pedigree.
genIndex	An integer representing the current generation index for which the data frame is being created.
idGen	A numeric vector containing the ID numbers to be assigned to individuals in the current generation.

Value

A data frame representing the initial structure for the individuals in the specified generation before any relationships (parental, spousal) are defined. The columns include family ID ('fam'), individual ID ('id'), generation number ('gen'), father's ID ('pat'), mother's ID ('mat'), spouse's ID ('spID'), and sex ('sex'), with NA values for paternal, maternal, and spouse IDs, and sex.

Examples

```
sizeGens <- c(3, 5, 4) # Example sizes for 3 generations
genIndex <- 2 # Creating data frame for the 2nd generation
idGen <- 101:105 # Example IDs for the 2nd generation
df_Ngen <- createGenDataFrame(sizeGens, genIndex, idGen)
print(df_Ngen)
```

determineSex	<i>Determine Sex of Offspring</i>
--------------	-----------------------------------

Description

This internal function assigns sexes to the offspring in a generation based on the specified sex ratio.

Usage

```
determineSex(idGen, sexR, code_male = "M", code_female = "F")
```

Arguments

- idGen Vector of IDs for the generation.
- sexR Numeric value indicating the sex ratio (proportion of males).
- code_male The value to use for males. Default is "M"
- code_female The value to use for females. Default is "F"

Value

Vector of sexes ("M" for male, "F" for female) for the offspring.

dropLink

*dropLink***Description**

A function to drop a person from his/her parents in the simulated pedigree data . frame. The person can be dropped by specifying his/her ID or by specifying the generation which the randomly to-be-dropped person is in. The function can separate one pedigree into two pedigrees. Separating into small pieces should be done by running the function multiple times. This is a supplementary function for simulatePedigree.

Usage

```
dropLink(
  ped,
  ID_drop = NA_integer_,
  gen_drop = 2,
  sex_drop = NA_character_,
  n_drop = 1
)
```

Arguments

ped	a pedigree simulated from simulatePedigree function or the same format
ID_drop	the ID of the person to be dropped from his/her parents.
gen_drop	the generation in which the randomly dropped person is. Will work if 'ID_drop' is not specified.
sex_drop	the biological sex of the randomly dropped person.
n_drop	the number of times the mutation happens.

Value

a pedigree with the dropped person's 'dadID' and 'momID' set to NA.

findBiggest

Function to find the biggest families in a pedigree This function finds the biggest families in a pedigree. It is supposed to be used internally by the summarize_pedigree function.

Description

Function to find the biggest families in a pedigree This function finds the biggest families in a pedigree. It is supposed to be used internally by the summarize_pedigree function.

Usage

```
findBiggest(foo_summary_dt, n_foorest = 5, n_foo_total = nrow(foo_summary_dt))
```

Arguments

`foo_summary_dt` A data.table containing the summary statistics.

`n_foorest` An integer specifying the number of individuals in the summary.

`n_foo_total` An integer specifying the total number of individuals in the summary.

Value

a data.table containing the biggest families in the pedigree.

findOldest	<i>Function to find the oldest individuals in a pedigree This function finds the oldest families in a pedigree. It is supposed to be used internally by the summarize_pedigree function.</i>
------------	--

Description

Function to find the oldest individuals in a pedigree This function finds the oldest families in a pedigree. It is supposed to be used internally by the summarize_pedigree function.

Usage

```
findOldest(
  foo_summary_dt,
  byr = "byr",
  n_foorest = 5,
  n_foo_total = nrow(foo_summary_dt)
)
```

Arguments

`foo_summary_dt` A data.table containing the summary statistics.

`byr` Character. Optional column name for birth year. Used to determine the oldest lineages.

`n_foorest` An integer specifying the number of individuals in the summary.

`n_foo_total` An integer specifying the total number of individuals in the summary.

Value

a data.table containing the oldest families in the pedigree.

fitComponentModel	<i>fitComponentModel Fit the estimated variance components of a model to covariance data</i>
-------------------	--

Description

fitComponentModel Fit the estimated variance components of a model to covariance data

Usage

```
fitComponentModel(covmat, ...)
```

Arguments

covmat	The covariance matrix of the raw data, which may be blockwise.
...	Comma-separated relatedness component matrices representing the variance components of the model.

Details

This function fits the estimated variance components of a model to given covariance data. The rank of the component matrices is checked to ensure that the variance components are all identified. Warnings are issued if there are inconsistencies.

Value

A regression (linear model fitted with `lm`). The coefficients of the regression represent the estimated variance components.

Examples

```
## Not run:
# install.packages("OpenMX")
data(twinData, package = "OpenMx")
sellVars <- c("ht1", "ht2")
mzData <- subset(twinData, zyg %in% c(1), c(selVars, "zyg"))
dzData <- subset(twinData, zyg %in% c(3), c(selVars, "zyg"))

fitComponentModel(
  covmat = list(cov(mzData[, selVars], use = "pair"), cov(dzData[, selVars], use = "pair")),
  A = list(matrix(1, nrow = 2, ncol = 2), matrix(c(1, 0.5, 0.5, 1), nrow = 2, ncol = 2)),
  C = list(matrix(1, nrow = 2, ncol = 2), matrix(1, nrow = 2, ncol = 2)),
  E = list(diag(1, nrow = 2), diag(1, nrow = 2))
)

## End(Not run)
```

hazard	<i>Simulated pedigree with two extended families and an age-related hazard</i>
--------	--

Description

A dataset simulated to have an age-related hazard. There are two extended families that are sampled from the same population.

Usage

```
data(hazard)
```

Format

A data frame with 43 rows and 14 variables

Details

The variables are as follows:

- FamID: ID of the extended family
- ID: Person identification variable
- sex: Sex of the ID: 1 is female; 0 is male
- dadID: ID of the father
- momID: ID of the mother
- affected: logical. Whether the person is affected or not
- DA1: Binary variable signifying the meaninglessness of life
- DA2: Binary variable signifying the fundamental unknowability of existence
- birthYr: Birth year for person
- onsetYr: Year of onset for person
- deathYr: Death year for person
- available: logical. Whether
- Gen: Generation of the person
- proband: logical. Whether the person is a proband or not

`identifyComponentModel`*identifyComponentModel Determine if a variance components model is identified*

Description

`identifyComponentModel` Determine if a variance components model is identified

Usage

```
identifyComponentModel(..., verbose = TRUE)
```

Arguments

<code>...</code>	Comma-separated relatedness component matrices representing the variance components of the model.
<code>verbose</code>	logical. If FALSE, suppresses messages about identification; TRUE by default.

Details

This function checks the identification status of a given variance components model by examining the rank of the concatenated matrices of the components. If any components are not identified, their names are returned in the output.

Value

A list of length 2 containing:

- `identified`: TRUE if the model is identified, FALSE otherwise.
- `nidp`: A vector of non-identified parameters, specifying the names of components that are not simultaneously identified.

Examples

```
identifyComponentModel(A = list(matrix(1, 2, 2)), C = list(matrix(1, 2, 2)), E = diag(1, 2))
```

inbreeding

*Artificial pedigree data on eight families with inbreeding***Description**

A dataset created purely from imagination that includes several types of inbreeding. Different kinds of inbreeding occur in each extended family.

The types of inbreeding are as follows:

- Extended Family 1: Sister wives - Children with the same father and different mothers who are sisters.
- Extended Family 2: Full siblings have children.
- Extended Family 3: Half siblings have children.
- Extended Family 4: First cousins have children.
- Extended Family 5: Father has child with his daughter.
- Extended Family 6: Half sister wives - Children with the same father and different mothers who are half sisters.
- Extended Family 7: Uncle-niece and Aunt-nephew have children.
- Extended Family 8: A father-son pairs has children with a corresponding mother-daughter pair.

Although not all of the above structures are technically inbreeding, they aim to test pedigree diagramming and path tracing algorithms. This dataset is not intended to represent any real individuals or families.

The variables are as follows:

- ID: Person identification variable
- sex: Sex of the ID: 1 is female; 0 is male
- dadID: ID of the father
- momID: ID of the mother
- FamID: ID of the extended family
- Gen: Generation of the person
- proband: Always FALSE

Usage

```
data(inbreeding)
```

Format

A data frame (and ped object) with 134 rows and 7 variables

initializeRecord	<i>Initialize an Empty Individual Record</i>
------------------	--

Description

Creates a named list with all GEDCOM initialized to NA_character_.

Usage

```
initializeRecord(all_var_names)
```

Arguments

all_var_names A character vector of variable names.

Value

A named list representing an empty individual record.

insertEven	<i>evenInsert A function to insert m elements evenly into a length n vector.</i>
------------	--

Description

evenInsert A function to insert m elements evenly into a length n vector.

Usage

```
insertEven(m, n, verbose = FALSE)
```

```
evenInsert(m, n, verbose = FALSE)
```

Arguments

m	A numeric vector of length less than or equal to n. The elements to be inserted.
n	A numeric vector. The vector into which the elements of m will be inserted.
verbose	logical If TRUE, prints additional information. Default is FALSE.

Details

The function takes two vectors, m and n, and inserts the elements of m evenly into n. If the length of m is greater than the length of n, the vectors are swapped, and the insertion proceeds. The resulting vector is a combination of m and n, with the elements of m evenly distributed within n.

Value

Returns a numeric vector with the elements of `m` evenly inserted into `n`.

See Also

[SimPed](#) for the main function that uses this supporting function.

<code>makeInbreeding</code>	<i>makeInbreeding</i>
-----------------------------	-----------------------

Description

A function to create inbred mates in the simulated pedigree data.frame. Inbred mates can be created by specifying their IDs or the generation the inbred mate should be created. When specifying the generation, inbreeding between siblings or 1st cousin needs to be specified. This is a supplementary function for `simulatePedigree`.

Usage

```
makeInbreeding(
  ped,
  ID_mate1 = NA_integer_,
  ID_mate2 = NA_integer_,
  verbose = FALSE,
  gen_inbred = 2,
  type_inbred = "sib"
)
```

Arguments

<code>ped</code>	A data.frame in the same format as the output of <code>simulatePedigree</code> .
<code>ID_mate1</code>	A vector of ID of the first mate. If not provided, the function will randomly select two individuals from the second generation.
<code>ID_mate2</code>	A vector of ID of the second mate.
<code>verbose</code>	logical. If TRUE, print progress through stages of algorithm
<code>gen_inbred</code>	A vector of generation of the twin to be imputed.
<code>type_inbred</code>	A character vector indicating the type of inbreeding. "sib" for sibling inbreeding and "cousin" for cousin inbreeding.

Details

This function creates inbred mates in the simulated pedigree data.frame. This function's purpose is to evaluate the effect of inbreeding on model fitting and parameter estimation. In case it needs to be said, we do not condone inbreeding in real life. But we recognize that it is a common practice in some fields to create inbred strains for research purposes.

Value

Returns a `data.frame` with some inbred mates.

<code>makeTwins</code>	<i>makeTwins</i>
------------------------	------------------

Description

A function to impute twins in the simulated pedigree `data.frame`. Twins can be imputed by specifying their IDs or by specifying the generation the twin should be imputed. This is a supplementary function for `simulatePedigree`.

Usage

```
makeTwins(  
  ped,  
  ID_twin1 = NA_integer_,  
  ID_twin2 = NA_integer_,  
  gen_twin = 2,  
  verbose = FALSE,  
  zygosity = "MZ"  
)
```

Arguments

<code>ped</code>	A <code>data.frame</code> in the same format as the output of <code>simulatePedigree</code> .
<code>ID_twin1</code>	A vector of ID of the first twin.
<code>ID_twin2</code>	A vector of ID of the second twin.
<code>gen_twin</code>	A vector of generation of the twin to be imputed.
<code>verbose</code>	logical. If TRUE, print progress through stages of algorithm
<code>zygosity</code>	A character string indicating the zygosity of the twins. Default is "MZ" for monozygotic twins.

Value

Returns a `data.frame` with MZ twins information added as a new column.

mapFAMS2parents	<i>Create a Mapping from Family IDs to Parent IDs</i>
-----------------	---

Description

This function scans the data frame and creates a mapping of family IDs to the corresponding parent IDs.

Usage

```
mapFAMS2parents(df_temp)
```

Arguments

`df_temp` A data frame produced by `readGedcom()`.

Value

A list mapping family IDs to parent information.

markPotentialChildren	<i>Mark and Assign children</i>
-----------------------	---------------------------------

Description

This subfunction marks individuals in a generation as potential sons, daughters, or parents based on their relationships and assigns unique couple IDs. It processes the assignment of roles and relationships within and between generations in a pedigree simulation.

Usage

```
markPotentialChildren(
  df_Ngen,
  i,
  Ngen,
  sizeGens,
  CoupleF,
  code_male = "M",
  code_female = "F"
)
```

Arguments

df_Ngen	A data frame for the current generation being processed. It must include columns for individual IDs ('id'), spouse IDs ('spID'), sex ('sex'), and any previously assigned roles ('ifparent', 'ifson', 'ifdau').
i	Integer, the index of the current generation being processed.
Ngen	Integer, the total number of generations in the simulation.
sizeGens	Numeric vector, containing the size (number of individuals) of each generation.
CoupleF	Integer, IT MIGHT BE the number of couples in the current generation.
code_male	The value to use for males. Default is "M"
code_female	The value to use for females. Default is "F"

Value

Modifies 'df_Ngen' in place by updating or adding columns related to individual roles ('ifparent', 'ifson', 'ifdau') and couple IDs ('coupleId'). The updated data frame is also returned for integration into the larger pedigree data frame ('df_Fam').

parseNameLine	<i>Parse Name Line</i>
---------------	------------------------

Description

Extracts full name information from a GEDCOM "NAME" line and updates the record accordingly.

Usage

```
parseNameLine(line, record)
```

Arguments

line	A character string containing the name line.
record	A named list representing the individual's record.

Value

The updated record with parsed name information.

ped2add

*Take a pedigree and turn it into an additive genetics relatedness matrix***Description**

Take a pedigree and turn it into an additive genetics relatedness matrix

Usage

```
ped2add(
  ped,
  max_gen = 25,
  sparse = TRUE,
  verbose = FALSE,
  gc = FALSE,
  flatten_diag = FALSE,
  standardize_colnames = TRUE,
  transpose_method = "tcrossprod",
  adjacency_method = "direct",
  saveable = FALSE,
  resume = FALSE,
  save_rate = 5,
  save_rate_gen = save_rate,
  save_rate_parlist = 1e+05 * save_rate,
  save_path = "checkpoint/",
  compress = TRUE,
  ...
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
max_gen	the maximum number of generations to compute (e.g., only up to 4th degree relatives). The default is 25. However it can be set to infinity. 'Inf' uses as many generations as there are in the data.
sparse	logical. If TRUE, use and return sparse matrices from Matrix package
verbose	logical. If TRUE, print progress through stages of algorithm
gc	logical. If TRUE, do frequent garbage collection via gc to save memory
flatten_diag	logical. If TRUE, overwrite the diagonal of the final relatedness matrix with ones
standardize_colnames	logical. If TRUE, standardize the column names of the pedigree dataset
transpose_method	character. The method to use for computing the transpose. Options are "tcrossprod", "crossprod", or "star"

adjacency_method	character. The method to use for computing the adjacency matrix. Options are "loop", "indexed", direct or beta
saveable	logical. If TRUE, save the intermediate results to disk
resume	logical. If TRUE, resume from a checkpoint
save_rate	numeric. The rate at which to save the intermediate results
save_rate_gen	numeric. The rate at which to save the intermediate results by generation. If NULL, defaults to save_rate
save_rate_parlist	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to save_rate*1000
save_path	character. The path to save the checkpoint files
compress	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.
...	additional arguments to be passed to ped2com

Details

The algorithms and methodologies used in this function are further discussed and exemplified in the vignette titled "examplePedigreeFunctions". For more advanced scenarios and detailed explanations, consult this vignette.

ped2ce	<i>Take a pedigree and turn it into an extended environmental relatedness matrix</i>
--------	--

Description

Take a pedigree and turn it into an extended environmental relatedness matrix

Usage

```
ped2ce(ped, ...)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
...	additional arguments to be passed to ped2com

Details

The algorithms and methodologies used in this function are further discussed and exemplified in the vignette titled "examplePedigreeFunctions". For more advanced scenarios and detailed explanations, consult this vignette.

ped2cn	<i>Take a pedigree and turn it into a common nuclear environmental matrix</i>
--------	---

Description

Take a pedigree and turn it into a common nuclear environmental matrix

Usage

```
ped2cn(
  ped,
  max_gen = 25,
  sparse = TRUE,
  verbose = FALSE,
  gc = FALSE,
  flatten_diag = FALSE,
  standardize_colnames = TRUE,
  transpose_method = "tcrossprod",
  saveable = FALSE,
  resume = FALSE,
  save_rate = 5,
  adjacency_method = "direct",
  save_rate_gen = save_rate,
  save_rate_parlist = 1000 * save_rate,
  save_path = "checkpoint/",
  compress = TRUE,
  ...
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
max_gen	the maximum number of generations to compute (e.g., only up to 4th degree relatives). The default is 25. However it can be set to infinity. ‘Inf’ uses as many generations as there are in the data.
sparse	logical. If TRUE, use and return sparse matrices from Matrix package
verbose	logical. If TRUE, print progress through stages of algorithm
gc	logical. If TRUE, do frequent garbage collection via gc to save memory
flatten_diag	logical. If TRUE, overwrite the diagonal of the final relatedness matrix with ones
standardize_colnames	logical. If TRUE, standardize the column names of the pedigree dataset
transpose_method	character. The method to use for computing the transpose. Options are "tcrossprod", "crossprod", or "star"

saveable	logical. If TRUE, save the intermediate results to disk
resume	logical. If TRUE, resume from a checkpoint
save_rate	numeric. The rate at which to save the intermediate results
adjacency_method	character. The method to use for computing the adjacency matrix. Options are "loop", "indexed", direct or beta
save_rate_gen	numeric. The rate at which to save the intermediate results by generation. If NULL, defaults to save_rate
save_rate_parlist	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to save_rate*1000
save_path	character. The path to save the checkpoint files
compress	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.
...	additional arguments to be passed to ped2com

Details

The algorithms and methodologies used in this function are further discussed and exemplified in the vignette titled "examplePedigreeFunctions". For more advanced scenarios and detailed explanations, consult this vignette.

ped2com

Take a pedigree and turn it into a relatedness matrix

Description

Take a pedigree and turn it into a relatedness matrix

Usage

```
ped2com(
  ped,
  component,
  max_gen = 25,
  sparse = TRUE,
  verbose = FALSE,
  gc = FALSE,
  flatten_diag = FALSE,
  standardize_colnames = TRUE,
  transpose_method = "tcrossprod",
  adjacency_method = "direct",
  isChild_method = "classic",
  saveable = FALSE,
  resume = FALSE,
```

```

    save_rate = 5,
    save_rate_gen = save_rate,
    save_rate_parlist = 1e+05 * save_rate,
    update_rate = 100,
    save_path = "checkpoint/",
    adjBeta_method = NULL,
    compress = TRUE,
    ...
)

```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
component	character. Which component of the pedigree to return. See Details.
max_gen	the maximum number of generations to compute (e.g., only up to 4th degree relatives). The default is 25. However it can be set to infinity. 'Inf' uses as many generations as there are in the data.
sparse	logical. If TRUE, use and return sparse matrices from Matrix package
verbose	logical. If TRUE, print progress through stages of algorithm
gc	logical. If TRUE, do frequent garbage collection via gc to save memory
flatten_diag	logical. If TRUE, overwrite the diagonal of the final relatedness matrix with ones
standardize_colnames	logical. If TRUE, standardize the column names of the pedigree dataset
transpose_method	character. The method to use for computing the transpose. Options are "tcrossprod", "crossprod", or "star"
adjacency_method	character. The method to use for computing the adjacency matrix. Options are "loop", "indexed", direct or beta
isChild_method	character. The method to use for computing the isChild matrix. Options are "classic" or "partialparent"
saveable	logical. If TRUE, save the intermediate results to disk
resume	logical. If TRUE, resume from a checkpoint
save_rate	numeric. The rate at which to save the intermediate results
save_rate_gen	numeric. The rate at which to save the intermediate results by generation. If NULL, defaults to save_rate
save_rate_parlist	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to save_rate*1000
update_rate	numeric. The rate at which to print progress
save_path	character. The path to save the checkpoint files
adjBeta_method	numeric. The method to use for computing the building the adjacency_method matrix when using the "beta" build

compress	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.
...	additional arguments to be passed to ped2com

Details

The algorithms and methodologies used in this function are further discussed and exemplified in the vignette titled "examplePedigreeFunctions". For more advanced scenarios and detailed explanations, consult this vignette.

ped2fam

Segment Pedigree into Extended Families

Description

This function adds an extended family ID variable to a pedigree by segmenting that dataset into independent extended families using the weakly connected components algorithm.

Usage

```
ped2fam(
  ped,
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  famID = "famID",
  ...
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
famID	character. Name of the column to be created in ped for the family ID variable
...	additional arguments to be passed to ped2com

Details

The general idea of this function is to use person ID, mother ID, and father ID to create an extended family ID such that everyone with the same family ID is in the same (perhaps very extended) pedigree. That is, a pair of people with the same family ID have at least one traceable relation of any length to one another.

This function works by turning the pedigree into a mathematical graph using the igraph package. Once in graph form, the function uses weakly connected components to search for all possible relationship paths that could connect anyone in the data to anyone else in the data.

Value

A pedigree dataset with one additional column for the newly created extended family ID

ped2gen	<i>Take a pedigree and turn it into a generation relatedness matrix</i>
---------	---

Description

Take a pedigree and turn it into a generation relatedness matrix

Usage

```
ped2gen(
  ped,
  max_gen = 25,
  sparse = TRUE,
  verbose = FALSE,
  gc = FALSE,
  flatten_diag = FALSE,
  standardize_colnames = TRUE,
  transpose_method = "tcrossprod",
  saveable = FALSE,
  resume = FALSE,
  save_rate = 5,
  adjacency_method = "direct",
  save_rate_gen = save_rate,
  save_rate_parlist = 1000 * save_rate,
  save_path = "checkpoint/",
  compress = TRUE,
  ...
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
max_gen	the maximum number of generations to compute (e.g., only up to 4th degree relatives). The default is 25. However it can be set to infinity. ‘Inf’ uses as many generations as there are in the data.
sparse	logical. If TRUE, use and return sparse matrices from Matrix package
verbose	logical. If TRUE, print progress through stages of algorithm
gc	logical. If TRUE, do frequent garbage collection via gc to save memory
flatten_diag	logical. If TRUE, overwrite the diagonal of the final relatedness matrix with ones
standardize_colnames	logical. If TRUE, standardize the column names of the pedigree dataset

transpose_method	character. The method to use for computing the transpose. Options are "tcrossprod", "crossprod", or "star"
saveable	logical. If TRUE, save the intermediate results to disk
resume	logical. If TRUE, resume from a checkpoint
save_rate	numeric. The rate at which to save the intermediate results
adjacency_method	character. The method to use for computing the adjacency matrix. Options are "loop", "indexed", direct or beta
save_rate_gen	numeric. The rate at which to save the intermediate results by generation. If NULL, defaults to save_rate
save_rate_parlist	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to save_rate*1000
save_path	character. The path to save the checkpoint files
compress	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.
...	additional arguments to be passed to ped2com

Details

The algorithms and methodologies used in this function are further discussed and exemplified in the vignette titled "examplePedigreeFunctions". For more advanced scenarios and detailed explanations, consult this vignette.

ped2graph	<i>Turn a pedigree into a graph</i>
-----------	-------------------------------------

Description

Turn a pedigree into a graph

Usage

```
ped2graph(
  ped,
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  directed = TRUE,
  adjacent = c("parents", "mothers", "fathers"),
  ...
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
directed	Logical scalar. Default is TRUE. Indicates whether or not to create a directed graph.
adjacent	Character. Relationship that defines adjacency in the graph: parents, mothers, or fathers
...	additional arguments to be passed to ped2com

Details

The general idea of this function is to represent a pedigree as a graph using the igraph package.

Once in graph form, several common pedigree tasks become much simpler.

The adjacent argument allows for different kinds of graph structures. When using parents for adjacency, the graph shows all parent-child relationships. When using mother for adjacency, the graph only shows mother-child relationships. Similarly when using father for adjacency, only father-child relationships appear in the graph. Construct extended families from the parent graph, maternal lines from the mothers graph, and paternal lines from the fathers graph.

Value

A graph

ped2maternal

Add a maternal line ID variable to a pedigree

Description

Add a maternal line ID variable to a pedigree

Usage

```
ped2maternal(
  ped,
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  matID = "matID",
  ...
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
matID	Character. Maternal line ID variable to be created and added to the pedigree
...	additional arguments to be passed to ped2com

Details

Under various scenarios it is useful to know which people in a pedigree belong to the same maternal lines. This function first turns a pedigree into a graph where adjacency is defined by mother-child relationships. Subsequently, the weakly connected components algorithm finds all the separate maternal lines and gives them an ID variable.

See Also

[[ped2fam\(\)](#)] for creating extended family IDs, and [[ped2paternal\(\)](#)] for creating paternal line IDs

ped2mit

Take a pedigree and turn it into a mitochondrial relatedness matrix

Description

Take a pedigree and turn it into a mitochondrial relatedness matrix

Usage

```
ped2mit(
  ped,
  max_gen = 25,
  sparse = TRUE,
  verbose = FALSE,
  gc = FALSE,
  flatten_diag = FALSE,
  standardize_colnames = TRUE,
  transpose_method = "tcrossprod",
  adjacency_method = "direct",
  saveable = FALSE,
  resume = FALSE,
  save_rate = 5,
  save_rate_gen = save_rate,
  save_rate_parlist = 1e+05 * save_rate,
  save_path = "checkpoint/",
  compress = TRUE,
  ...
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
max_gen	the maximum number of generations to compute (e.g., only up to 4th degree relatives). The default is 25. However it can be set to infinity. 'Inf' uses as many generations as there are in the data.
sparse	logical. If TRUE, use and return sparse matrices from Matrix package
verbose	logical. If TRUE, print progress through stages of algorithm
gc	logical. If TRUE, do frequent garbage collection via gc to save memory
flatten_diag	logical. If TRUE, overwrite the diagonal of the final relatedness matrix with ones
standardize_colnames	logical. If TRUE, standardize the column names of the pedigree dataset
transpose_method	character. The method to use for computing the transpose. Options are "tcrossprod", "crossprod", or "star"
adjacency_method	character. The method to use for computing the adjacency matrix. Options are "loop", "indexed", direct or beta
saveable	logical. If TRUE, save the intermediate results to disk
resume	logical. If TRUE, resume from a checkpoint
save_rate	numeric. The rate at which to save the intermediate results
save_rate_gen	numeric. The rate at which to save the intermediate results by generation. If NULL, defaults to save_rate
save_rate_parlist	numeric. The rate at which to save the intermediate results by parent list. If NULL, defaults to save_rate*1000
save_path	character. The path to save the checkpoint files
compress	logical. If TRUE, use compression when saving the checkpoint files. Defaults to TRUE.
...	additional arguments to be passed to ped2com

Details

The algorithms and methodologies used in this function are further discussed and exemplified in the vignette titled "examplePedigreeFunctions". For more advanced scenarios and detailed explanations, consult this vignette.

ped2paternal	<i>Add a paternal line ID variable to a pedigree</i>
--------------	--

Description

Add a paternal line ID variable to a pedigree

Usage

```
ped2paternal(  
  ped,  
  personID = "ID",  
  momID = "momID",  
  dadID = "dadID",  
  patID = "patID",  
  ...  
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
patID	Character. Paternal line ID variable to be created and added to the pedigree
...	additional arguments to be passed to ped2com

Details

Under various scenarios it is useful to know which people in a pedigree belong to the same paternal lines. This function first turns a pedigree into a graph where adjacency is defined by father-child relationships. Subsequently, the weakly connected components algorithm finds all the separate paternal lines and gives them an ID variable.

See Also

[ped2fam()] for creating extended family IDs, and [ped2maternal()] for creating maternal line IDs

postProcessGedcom	<i>Post-process GEDCOM Data Frame</i>
-------------------	---------------------------------------

Description

This function optionally adds parent information, combines duplicate columns, and removes empty columns from the GEDCOM data frame. It is called by `readGedcom()` if `post_process = TRUE`.

Usage

```
postProcessGedcom(
  df_temp,
  remove_empty_cols = TRUE,
  combine_cols = TRUE,
  add_parents = TRUE,
  skinny = TRUE,
  verbose = FALSE
)
```

Arguments

<code>df_temp</code>	A data frame produced by <code>readGedcom()</code> .
<code>remove_empty_cols</code>	Logical indicating whether to remove columns that are entirely missing.
<code>combine_cols</code>	Logical indicating whether to combine columns with duplicate values.
<code>add_parents</code>	Logical indicating whether to add parent information.
<code>skinny</code>	Logical indicating whether to slim down the data frame.
<code>verbose</code>	Logical indicating whether to print progress messages.

Value

The post-processed data frame.

potter	<i>Fictional pedigree data on a wizarding family</i>
--------	--

Description

A dataset created for educational and illustrative use, containing a fictional pedigree modeled after characters from the Harry Potter series. This data is structured for use in software demonstrations involving pedigree diagrams, inheritance structures, and kinship modeling. This dataset is not intended to represent any real individuals or families. It includes no narrative content or protected expression from the original works and is provided solely for educational purposes. This dataset is not endorsed by or affiliated with the creators or copyright holders of the Harry Potter series.

Usage

```
data(potter)
```

Format

A data frame (and ped object) with 36 rows and 10 variables

Details

The variables are as follows:

- personID: Person identification variable
- famID: Family identification variable
- name: Name of the person
- first_name: First name of the person
- surname: Last name of the person
- gen: Generation of the person
- momID: ID of the mother
- dadID: ID of the father
- spouseID: ID of the spouse
- sex: Sex of the ID: 1 is male; 0 is female
- twinID: ID of the twin, if applicable
- zygosity: Zygosity of the twin, if applicable. mz is monozygotic; dz is dizygotic

IDs in the 100s momIDs and dadIDs are for people not in the dataset.

prepSummarizePedigrees

Function to prepare the pedigree for summarization This function prepares the pedigree for summarization by ensuring that the necessary IDs are present and that the pedigree is built correctly.

Description

Function to prepare the pedigree for summarization This function prepares the pedigree for summarization by ensuring that the necessary IDs are present and that the pedigree is built correctly.

Usage

```
prepSummarizePedigrees(  
  ped,  
  type,  
  verbose = FALSE,  
  famID,  
  personID,  
  momID,  
  dadID,  
  matID,  
  patID  
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
type	Character vector. Specifies which summaries to compute. Options: "fathers", "mothers", "families". Default includes all three.
verbose	Logical, if TRUE, print progress messages.
famID	character. Name of the column to be created in ped for the family ID variable
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
matID	Character. Maternal line ID variable to be created and added to the pedigree
patID	Character. Paternal line ID variable to be created and added to the pedigree

processEventLine	<i>Process Event Lines (Birth or Death)</i>
------------------	---

Description

Extracts event details (e.g., date, place, cause, latitude, longitude) from a block of GEDCOM lines. For "birth": expect DATE on line i+1, PLAC on i+2, LATI on i+4, LONG on i+5. For "death": expect DATE on line i+1, PLAC on i+2, CAUS on i+3, LATI on i+4, LONG on i+5.

Usage

```
processEventLine(event, block, i, record, pattern_rows)
```

Arguments

event	A character string indicating the event type ("birth" or "death").
block	A character vector of GEDCOM lines.
i	The current line index where the event tag is found.
record	A named list representing the individual's record.
pattern_rows	A list with counts of GEDCOM tag occurrences.

Value

The updated record with parsed event information.#

processParents	<i>Process Parents Information from GEDCOM Data</i>
----------------	---

Description

This function adds mother and father IDs to individuals in the data frame

Usage

```
processParents(df_temp, datasource)
```

Arguments

df_temp	A data frame produced by readGedcom().
datasource	Character string indicating the data source ("gedcom" or "wiki").

Value

The updated data frame with parent IDs added.

readGedcom	<i>Read a GEDCOM File</i>
------------	---------------------------

Description

This function ingests a GEDCOM genealogy file, identifies each individual described in the file, and parses their information into a structured data frame. It supports optional post-processing to enrich the raw data, such as inferring parental IDs, merging redundant name fields, and dropping uninformative columns.

Usage

```
readGedcom(
  file_path,
  verbose = FALSE,
  add_parents = TRUE,
  remove_empty_cols = TRUE,
  combine_cols = TRUE,
  skinny = FALSE,
  update_rate = 1000,
  post_process = TRUE,
  ...
)
```

```

)

readGed(
  file_path,
  verbose = FALSE,
  add_parents = TRUE,
  remove_empty_cols = TRUE,
  combine_cols = TRUE,
  skinny = FALSE,
  update_rate = 1000,
  post_process = TRUE,
  ...
)

readgedcom(
  file_path,
  verbose = FALSE,
  add_parents = TRUE,
  remove_empty_cols = TRUE,
  combine_cols = TRUE,
  skinny = FALSE,
  update_rate = 1000,
  post_process = TRUE,
  ...
)

```

Arguments

<code>file_path</code>	Character. Path to the GEDCOM file.
<code>verbose</code>	Logical. If TRUE, print progress messages.
<code>add_parents</code>	Logical. If TRUE, add momID and dadID via FAMC/FAMS mapping.
<code>remove_empty_cols</code>	Logical. If TRUE, drop columns that are entirely NA.
<code>combine_cols</code>	Logical. If TRUE, merge duplicate name columns (e.g., given/surn pieces).
<code>skinny</code>	Logical. If TRUE, return a slimmer data frame (drops FAMC, FAMS and all-empty cols).
<code>update_rate</code>	Numeric. Intended rate at which to print progress
<code>post_process</code>	Logical. If TRUE, apply post-processing (parents, combine, drop empty, skinny).
<code>...</code>	Additional arguments to be passed to the function.

Details

The parser operates line-by-line and is tuned to the common GEDCOM 5.5/5.5.1 structure: This parser is line-oriented. Individuals are defined by blocks that start with a line containing "@ INDI". Within each block, tags are parsed using simple pattern matches: - Relationship tags FAMC (as child) and FAMS (as spouse) are collected and later mapped to parent IDs if `add_parents = TRUE`.

- Individuals are defined in blocks beginning with lines containing @ INDI. Each block is passed to an internal parser that extracts identifiers, names, life events, attributes, and family relationships.
- Names are parsed from the GEDCOM NAME tag, which usually encodes the given name and surname with slashes (e.g., "NAME John /Smith/"). The parser extracts the given name, surname, and constructs a cleaned full name. Additional name components (prefix, suffix, nickname, married surname) are parsed if present.
- Life events are recognized by BIRT and DEAT tags. Event details are assumed to occur at fixed offsets in the block (for example, a BIRT tag is followed by a DATE, then a PLAC, and optionally geographic coordinates). Missing elements leave the corresponding field as NA. for birth, expected lines are DATE (i+1), PLAC (i+2), LATI (i+4), LONG (i+5); for death, expected lines are DATE (i+1), PLAC (i+2), CAUS (i+3), LATI (i+4), LONG (i+5).
- Attributes such as occupation, education, and religion are parsed directly from GEDCOM tags (OCCU, EDUC, RELI, etc.). Each attribute is stored in a dedicated column prefixed with attribute_.
- Relationships are parsed from FAMC (family as child) and FAMS (family as spouse). These identifiers are preserved in the raw output and can optionally be mapped to explicit parent IDs via processParents().
- Post-processing can be applied by setting post_process = TRUE. This applies several clean-up steps: adding inferred parents, merging duplicate name fields, and slimming the data frame by removing all-empty columns or relationship tags.

Value

A data frame containing information about individuals, with the following potential columns: - 'personID': ID of the individual parsed from the @ INDI line - 'momID': ID of the individual's mother - 'dadID': ID of the individual's father - 'sex': Sex of the individual - 'name': Full name of the individual - 'name_given': First name of the individual - 'name_surn': Last name of the individual - 'name_marriedsurn': Married name of the individual - 'name_nick': Nickname of the individual - 'name_npfx': Name prefix - 'name_nsfx': Name suffix - 'birth_date': Birth date of the individual - 'birth_lat': Latitude of the birthplace - 'birth_long': Longitude of the birthplace - 'birth_place': Birthplace of the individual - 'death_caus': Cause of death - 'death_date': Death date of the individual - 'death_lat': Latitude of the place of death - 'death_long': Longitude of the place of death - 'death_place': Place of death of the individual - 'attribute_caste': Caste of the individual - 'attribute_children': Number of children of the individual - 'attribute_description': Description of the individual - 'attribute_education': Education of the individual - 'attribute_idnumber': Identification number of the individual - 'attribute_marriages': Number of marriages of the individual - 'attribute_nationality': Nationality of the individual - 'attribute_occupation': Occupation of the individual - 'attribute_property': Property owned by the individual - 'attribute_religion': Religion of the individual - 'attribute_residence': Residence of the individual - 'attribute_ssn': Social security number of the individual - 'attribute_title': Title of the individual - 'FAMC': ID(s) of the family where the individual is a child - 'FAMS': ID(s) of the family where the individual is a spouse

Description

Read Wiki Family Tree

Usage

```
readWikifamilytree(text = NULL, verbose = FALSE, file_path = NULL, ...)
```

Arguments

text	A character string containing the text of a family tree in wiki format.
verbose	A logical value indicating whether to print messages.
file_path	The path to the file containing the family tree.
...	Additional arguments (not used).

Value

A list containing the summary, members, structure, and relationships of the family tree.

recodeSex

Recodes Sex Variable in a Pedigree Dataframe

Description

This function serves as is primarily used internally, by plotting functions etc. It sets the 'repair' flag to TRUE automatically and forwards any additional parameters to 'checkSex'.

Usage

```
recodeSex(
  ped,
  verbose = FALSE,
  code_male = NULL,
  code_na = NULL,
  code_female = NULL,
  code_unknown = NULL,
  recode_male = "M",
  recode_female = "F",
  recode_unknown = "U",
  recode_na = NA_character_
)
```

Arguments

<code>ped</code>	A dataframe representing the pedigree data with a 'sex' column.
<code>verbose</code>	A logical flag indicating whether to print progress and validation messages to the console.
<code>code_male</code>	The current code used to represent males in the 'sex' column.
<code>code_na</code>	The current value used for missing values.
<code>code_female</code>	The current code used to represent females in the 'sex' column. If both are NULL, no recoding is performed.
<code>code_unknown</code>	The current code used to represent unknown or ambiguous sex in the 'sex' column. Can be NA to indicate that missing values should be treated as unknown. If NULL and both <code>code_male</code> and <code>code_female</code> are provided, values not matching either will be inferred as unknown.
<code>recode_male</code>	The value to use for males. Default is "M"
<code>recode_female</code>	The value to use for females. Default is "F"
<code>recode_unknown</code>	The value to use for unknown values. Default is "U"
<code>recode_na</code>	The value to use for missing values. Default is NA_character_

Details

The validation process identifies: - The unique sex codes present in the dataset. - Whether individuals listed as fathers or mothers have inconsistent sex codes. - Instances where an individual's recorded sex does not align with their parental role.

If `'repair = TRUE'`, the function standardizes sex coding by: - Assigning individuals listed as fathers the most common male code in the dataset. - Assigning individuals listed as mothers the most common female code.

This function uses the terms 'male' and 'female' in a biological context, referring to chromosomal and other biologically-based characteristics necessary for constructing genetic pedigrees. The biological aspect of sex used in genetic analysis (genotype) is distinct from the broader, richer concept of gender identity (phenotype).

We recognize the importance of using language and methodologies that affirm and respect the full spectrum of gender identities. The developers of this package express unequivocal support for folx in the transgender and LGBTQ+ communities.

Value

A modified version of the input data.frame `ped`, containing an additional or modified `'sex_recode'` column where the 'sex' values are recoded according to `code_male`. NA values in the 'sex' column are preserved.

repairIDs	<i>Repair Missing IDs</i>
-----------	---------------------------

Description

This function repairs missing IDs in a pedigree.

Usage

```
repairIDs(ped, verbose = FALSE)
```

Arguments

ped	A dataframe representing the pedigree data with columns 'ID', 'dadID', and 'momID'.
verbose	A logical flag indicating whether to print progress and validation messages to the console.

Value

A corrected pedigree

repairParentIDs	<i>Repair Parent IDs</i>
-----------------	--------------------------

Description

This function repairs parent IDs in a pedigree.

Usage

```
repairParentIDs(
  ped,
  verbose = FALSE,
  famID = "famID",
  personID = "ID",
  momID = "momID",
  dadID = "dadID"
)
```


Arguments

ped	A dataframe representing the pedigree data with columns 'ID', 'dadID', and 'momID'.
verbose	A logical flag indicating whether to print progress and validation messages to the console.
famID	Character. Column name for family IDs.
personID	Character. Column name for individual IDs.
momID	Character. Column name for maternal IDs.
dadID	Character. Column name for paternal IDs.

Value

A corrected pedigree

repairSex	<i>Repairs Sex Coding in a Pedigree Dataframe</i>
-----------	---

Description

This function serves as a wrapper around 'checkSex' to specifically handle the repair of the sex coding in a pedigree dataframe.

Usage

```
repairSex(
  ped,
  verbose = FALSE,
  code_male = NULL,
  code_female = NULL,
  code_unknown = NULL
)
```

Arguments

ped	A dataframe representing the pedigree data with a 'sex' column.
verbose	A logical flag indicating whether to print progress and validation messages to the console.
code_male	The current code used to represent males in the 'sex' column.
code_female	The current code used to represent females in the 'sex' column. If both are NULL, no recoding is performed.
code_unknown	The current code used to represent unknown or ambiguous sex in the 'sex' column. Can be NA to indicate that missing values should be treated as unknown. If NULL and both code_male and code_female are provided, values not matching either will be inferred as unknown.

Details

The validation process identifies:

- The unique sex codes present in the dataset.
- Whether individuals listed as fathers or mothers have inconsistent sex codes.
- Instances where an individual's recorded sex does not align with their parental role.

If 'repair = TRUE', the function standardizes sex coding by:

- Assigning individuals listed as fathers the most common male code in the dataset.
- Assigning individuals listed as mothers the most common female code.

This function uses the terms 'male' and 'female' in a biological context, referring to chromosomal and other biologically-based characteristics necessary for constructing genetic pedigrees. The biological aspect of sex used in genetic analysis (genotype) is distinct from the broader, richer concept of gender identity (phenotype).

We recognize the importance of using language and methodologies that affirm and respect the full spectrum of gender identities. The developers of this package express unequivocal support for folk in the transgender and LGBTQ+ communities.

Value

A modified version of the input data.frame ped, containing an additional or modified 'sex_recode' column where the 'sex' values are recoded according to code_male. NA values in the 'sex' column are preserved.

See Also

[checkSex](#)

Examples

```
## Not run:
ped <- data.frame(ID = c(1, 2, 3), sex = c("M", "F", "M"))
repairSex(ped, code_male = "M", verbose = TRUE)

## End(Not run)
```

royal92

Royal pedigree data from 1992

Description

A dataset created by Denis Reid from the Royal Families of Europe.

Usage

```
data(royal92)
```

Format

A data frame with 3110 observations

Details

The variables are as follows:

- id: Person identification variable
- momID: ID of the mother
- dadID: ID of the father
- name: Name of the person
- sex: Biological sex
- birth_date: Date of birth
- death_date: Date of death
- attribute_title: Title of the person

simulatePedigree	<i>Simulate Pedigrees This function simulates "balanced" pedigrees based on a group of parameters: 1) k - Kids per couple; 2) G - Number of generations; 3) p - Proportion of males in offspring; 4) r - Mating rate.</i>
------------------	---

Description

Simulate Pedigrees This function simulates "balanced" pedigrees based on a group of parameters: 1) k - Kids per couple; 2) G - Number of generations; 3) p - Proportion of males in offspring; 4) r - Mating rate.

Usage

```
simulatePedigree(
  kpc = 3,
  Ngen = 4,
  sexR = 0.5,
  marR = 2/3,
  rd_kpc = FALSE,
  balancedSex = TRUE,
  balancedMar = TRUE,
  verbose = FALSE,
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  spouseID = "spouseID",
  code_male = "M",
  code_female = "F",
  fam_shift = 1L
)

SimPed(...)
```

Arguments

kpc	Number of kids per couple. An integer ≥ 2 that determines how many kids each fertilized mated couple will have in the pedigree. Default value is 3. Returns an error when kpc equals 1.
Ngen	Number of generations. An integer ≥ 2 that determines how many generations the simulated pedigree will have. The first generation is always a fertilized couple. The last generation has no mated individuals.
sexR	Sex ratio of offspring. A numeric value ranging from 0 to 1 that determines the proportion of males in all offspring in this pedigree. For instance, 0.4 means 40 percent of the offspring will be male.
marR	Mating rate. A numeric value ranging from 0 to 1 which determines the proportion of mated (fertilized) couples in the pedigree within each generation. For instance, marR = 0.5 suggests 50 percent of the offspring in a specific generation will be mated and have their offspring.
rd_kpc	logical. If TRUE, the number of kids per mate will be randomly generated from a poisson distribution with mean kpc. If FALSE, the number of kids per mate will be fixed at kpc.
balancedSex	Not fully developed yet. Always TRUE in the current version.
balancedMar	Not fully developed yet. Always TRUE in the current version.
verbose	logical If TRUE, message progress through stages of algorithm
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
spouseID	The name of the column that will contain the spouse ID in the output data frame. Default is "spID".
code_male	The value to use for males. Default is "M"
code_female	The value to use for females. Default is "F"
fam_shift	An integer to shift the person ID. Default is 1L.
...	Additional arguments to be passed to other functions.

Value

A data.frame with each row representing a simulated individual. The columns are as follows:

- fam: The family id of each simulated individual. It is 'fam1' in a single simulated pedigree.
- ID: The unique personal ID of each simulated individual. The first digit is the fam id; the fourth digit is the generation the individual is in; the following digits represent the order of the individual within their pedigree. For example, 100411 suggests this individual has a family id of 1, is in the 4th generation, and is the 11th individual in the 4th generation.
- gen: The generation the simulated individual is in.
- dadID: Personal ID of the individual's father.
- momID: Personal ID of the individual's mother.
- spID: Personal ID of the individual's mate.
- sex: Biological sex of the individual. F - female; M - male.

Examples

```
set.seed(5)
df_ped <- simulatePedigree(
  kpc = 4,
  Ngen = 4,
  sexR = .5,
  marR = .7
)
summary(df_ped)
```

sliceFamilies

sliceFamilies

Description

Slices up families by additive relatedness, creating CSV files grouped by degree of relatedness. Operates on a potentially large file by reading in chunks and binning links by additive relatedness.

Usage

```
sliceFamilies(
  outcome_name = "AD_demo",
  biggest = TRUE,
  bin_width = 0.1,
  degreerelatedness = 12,
  chunk_size = 2e+07,
  max_lines = 1e+13,
  addRel_ceiling = 1.5,
  input_file = NULL,
  folder_prefix = "data",
  progress_csv = "progress.csv",
  progress_status = "progress.txt",
  data_directory = NULL,
  verbose = FALSE,
  error_handling = FALSE,
  file_column_names = c("ID1", "ID2", "addRel", "mitRel", "cnuRel")
)
```

Arguments

outcome_name	Name of the outcome variable (used for naming input/output files)
biggest	Logical; whether to process the "biggest" family dataset (TRUE) or all-but-biggest (FALSE)
bin_width	Width of additive relatedness bins (default is 0.10)
degreerelatedness	Maximum degree of relatedness to consider (default 12)

chunk_size	Number of lines to read in each chunk (default 2e7)
max_lines	Max number of lines to process from input file (default 1e13)
addRel_ceiling	Numeric. Maximum relatedness value to bin to. Default is 1.5
input_file	Path to the input CSV file. If NULL, defaults to a specific file based on 'biggest' flag.
folder_prefix	Prefix for the output folder (default "data")
progress_csv	Path to a CSV file for tracking progress (default "progress.csv")
progress_status	Path to a text file for logging progress status (default "progress.txt")
data_directory	Directory where output files will be saved. If NULL, it is constructed based on 'outcome_name' and 'folder_prefix'.
verbose	Logical; whether to print progress messages (default FALSE)
error_handling	Logical. Should more aggressive error handling be attempted? Default is false
file_column_names	Names of the columns in the input file (default c("ID1", "ID2", "addRel", "mitRel", "cnuRel"))

Value

NULL. Writes CSV files to disk and updates progress logs.

summarizeFamilies	<i>Summarize the families in a pedigree</i>
-------------------	---

Description

Summarize the families in a pedigree

Usage

```
summarizeFamilies(
  ped,
  famID = "famID",
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  matID = "matID",
  patID = "patID",
  byr = NULL,
  founder_sort_var = NULL,
  include_founder = FALSE,
  n_biggest = 5,
  n_oldest = 5,
  skip_var = NULL,
```

```

    five_num_summary = FALSE,
    verbose = FALSE,
    network_checks = FALSE
  )

  summariseFamilies(
    ped,
    famID = "famID",
    personID = "ID",
    momID = "momID",
    dadID = "dadID",
    matID = "matID",
    patID = "patID",
    byr = NULL,
    founder_sort_var = NULL,
    include_founder = FALSE,
    n_biggest = 5,
    n_oldest = 5,
    skip_var = NULL,
    five_num_summary = FALSE,
    verbose = FALSE,
    network_checks = FALSE
  )

```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
famID	character. Name of the column to be created in ped for the family ID variable
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
matID	Character. Maternal line ID variable to be created and added to the pedigree
patID	Character. Paternal line ID variable to be created and added to the pedigree
byr	Character. Optional column name for birth year. Used to determine the oldest lineages.
founder_sort_var	Character. Column used to determine the founder of each lineage. Defaults to 'byr' (if available) or 'personID' otherwise.
include_founder	Logical. If 'TRUE', includes the founder (originating member) of each lineage in the output.
n_biggest	Integer. Number of largest lineages to return (sorted by count).
n_oldest	Integer. Number of oldest lineages to return (sorted by birth year).
skip_var	Character vector. Variables to exclude from summary calculations.

five_num_summary Logical. If 'TRUE', includes the first quartile (Q1) and third quartile (Q3) in addition to the minimum, median, and maximum values.

verbose Logical, if TRUE, print progress messages.

network_checks Logical. If 'TRUE', performs network checks on the pedigree data.

See Also

[summarizePedigrees ()]

summarizeMatrilines	<i>Summarize the maternal lines in a pedigree</i>
---------------------	---

Description

Summarize the maternal lines in a pedigree

Usage

```
summarizeMatrilines(
  ped,
  famID = "famID",
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  matID = "matID",
  patID = "patID",
  byr = NULL,
  include_founder = FALSE,
  founder_sort_var = NULL,
  n_biggest = 5,
  n_oldest = 5,
  skip_var = NULL,
  five_num_summary = FALSE,
  verbose = FALSE,
  network_checks = FALSE
)
```

```
summariseMatrilines(
  ped,
  famID = "famID",
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  matID = "matID",
  patID = "patID",
  byr = NULL,
```



```

    include_founder = FALSE,
    founder_sort_var = NULL,
    n_biggest = 5,
    n_oldest = 5,
    skip_var = NULL,
    five_num_summary = FALSE,
    verbose = FALSE,
    network_checks = FALSE
  )

```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
famID	character. Name of the column to be created in ped for the family ID variable
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
matID	Character. Maternal line ID variable to be created and added to the pedigree
patID	Character. Paternal line ID variable to be created and added to the pedigree
byr	Character. Optional column name for birth year. Used to determine the oldest lineages.
include_founder	Logical. If 'TRUE', includes the founder (originating member) of each lineage in the output.
founder_sort_var	Character. Column used to determine the founder of each lineage. Defaults to 'byr' (if available) or 'personID' otherwise.
n_biggest	Integer. Number of largest lineages to return (sorted by count).
n_oldest	Integer. Number of oldest lineages to return (sorted by birth year).
skip_var	Character vector. Variables to exclude from summary calculations.
five_num_summary	Logical. If 'TRUE', includes the first quartile (Q1) and third quartile (Q3) in addition to the minimum, median, and maximum values.
verbose	Logical, if TRUE, print progress messages.
network_checks	Logical. If 'TRUE', performs network checks on the pedigree data.

See Also

[summarizePedigrees ()]

summarizePatriline	<i>Summarize the paternal lines in a pedigree</i>
--------------------	---

Description

Summarize the paternal lines in a pedigree

Usage

```
summarizePatriline(  
  ped,  
  famID = "famID",  
  personID = "ID",  
  momID = "momID",  
  dadID = "dadID",  
  matID = "matID",  
  patID = "patID",  
  byr = NULL,  
  founder_sort_var = NULL,  
  include_founder = FALSE,  
  n_biggest = 5,  
  n_oldest = 5,  
  skip_var = NULL,  
  five_num_summary = FALSE,  
  verbose = FALSE,  
  network_checks = FALSE  
)
```

```
summarisePatriline(  
  ped,  
  famID = "famID",  
  personID = "ID",  
  momID = "momID",  
  dadID = "dadID",  
  matID = "matID",  
  patID = "patID",  
  byr = NULL,  
  founder_sort_var = NULL,  
  include_founder = FALSE,  
  n_biggest = 5,  
  n_oldest = 5,  
  skip_var = NULL,  
  five_num_summary = FALSE,  
  verbose = FALSE,  
  network_checks = FALSE  
)
```

Arguments

ped	a pedigree dataset. Needs ID, momID, and dadID columns
famID	character. Name of the column to be created in ped for the family ID variable
personID	character. Name of the column in ped for the person ID variable
momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
matID	Character. Maternal line ID variable to be created and added to the pedigree
patID	Character. Paternal line ID variable to be created and added to the pedigree
byr	Character. Optional column name for birth year. Used to determine the oldest lineages.
founder_sort_var	Character. Column used to determine the founder of each lineage. Defaults to 'byr' (if available) or 'personID' otherwise.
include_founder	Logical. If 'TRUE', includes the founder (originating member) of each lineage in the output.
n_biggest	Integer. Number of largest lineages to return (sorted by count).
n_oldest	Integer. Number of oldest lineages to return (sorted by birth year).
skip_var	Character vector. Variables to exclude from summary calculations.
five_num_summary	Logical. If 'TRUE', includes the first quartile (Q1) and third quartile (Q3) in addition to the minimum, median, and maximum values.
verbose	Logical, if TRUE, print progress messages.
network_checks	Logical. If 'TRUE', performs network checks on the pedigree data.

See Also

[summarizePedigrees ()]

summarizePedigrees	<i>Summarize Pedigree Data</i>
--------------------	--------------------------------

Description

This function summarizes pedigree data, by computing key summary statistics for all numeric variables and identifying the originating member (founder) for each family, maternal, and paternal lineage.

Usage

```

summarizePedigrees(
  ped,
  famID = "famID",
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  matID = "matID",
  patID = "patID",
  type = c("fathers", "mothers", "families"),
  byr = NULL,
  include_founder = FALSE,
  founder_sort_var = NULL,
  n_keep = 5,
  n_biggest = n_keep,
  n_oldest = n_keep,
  skip_var = NULL,
  five_num_summary = FALSE,
  network_checks = FALSE,
  verbose = FALSE
)

```

```

summarisePedigrees(
  ped,
  famID = "famID",
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  matID = "matID",
  patID = "patID",
  type = c("fathers", "mothers", "families"),
  byr = NULL,
  include_founder = FALSE,
  founder_sort_var = NULL,
  n_keep = 5,
  n_biggest = n_keep,
  n_oldest = n_keep,
  skip_var = NULL,
  five_num_summary = FALSE,
  network_checks = FALSE,
  verbose = FALSE
)

```

Arguments

<code>ped</code>	a pedigree dataset. Needs ID, momID, and dadID columns
<code>famID</code>	character. Name of the column to be created in ped for the family ID variable
<code>personID</code>	character. Name of the column in ped for the person ID variable

momID	character. Name of the column in ped for the mother ID variable
dadID	character. Name of the column in ped for the father ID variable
matID	Character. Maternal line ID variable to be created and added to the pedigree
patID	Character. Paternal line ID variable to be created and added to the pedigree
type	Character vector. Specifies which summaries to compute. Options: "fathers", "mothers", "families". Default includes all three.
byr	Character. Optional column name for birth year. Used to determine the oldest lineages.
include_founder	Logical. If 'TRUE', includes the founder (originating member) of each lineage in the output.
founder_sort_var	Character. Column used to determine the founder of each lineage. Defaults to 'byr' (if available) or 'personID' otherwise.
n_keep	Integer. Number of lineages to keep in the output for each type of summary.
n_biggest	Integer. Number of largest lineages to return (sorted by count).
n_oldest	Integer. Number of oldest lineages to return (sorted by birth year).
skip_var	Character vector. Variables to exclude from summary calculations.
five_num_summary	Logical. If 'TRUE', includes the first quartile (Q1) and third quartile (Q3) in addition to the minimum, median, and maximum values.
network_checks	Logical. If 'TRUE', performs network checks on the pedigree data.
verbose	Logical, if TRUE, print progress messages.

Details

The function calculates standard descriptive statistics, including the count of individuals in each lineage, means, medians, minimum and maximum values, and standard deviations. Additionally, if 'five_num_summary = TRUE', the function includes the first and third quartiles (Q1, Q3) to provide a more detailed distributional summary. Users can also specify variables to exclude from the analysis via 'skip_var'.

Beyond summary statistics, the function identifies the founding member of each lineage based on the specified sorting variable ('founder_sort_var'), defaulting to birth year ('byr') when available or 'personID' otherwise. Users can retrieve the largest and oldest lineages by setting 'n_oldest' and 'n_biggest', respectively.

Value

A data.frame (or list) containing summary statistics for family, maternal, and paternal lines, as well as the 5 oldest and biggest lines.

traceTreePaths	<i>Trace paths between individuals in a family tree grid</i>
----------------	--

Description

Trace paths between individuals in a family tree grid

Usage

```
traceTreePaths(tree_long, deduplicate = TRUE)
```

Arguments

tree_long	A data.frame with columns: Row, Column, Value, id
deduplicate	Logical, if TRUE, will remove duplicate paths

Value

A data.frame with columns: from_id, to_id, direction, path_length, intermediates

validate_and_convert_matrix	<i>validate_and_convert_matrix</i>
-----------------------------	------------------------------------

Description

This function validates and converts a matrix to a specific format.

Usage

```
validate_and_convert_matrix(
  mat,
  name,
  ensure_symmetric = FALSE,
  force_binary = FALSE
)
```

Arguments

mat	The matrix to be validated and converted.
name	The name of the matrix for error messages.
ensure_symmetric	Logical indicating whether to ensure the matrix is symmetric.
force_binary	Logical indicating whether to force the matrix to be binary.

Value

The validated and converted matrix.

vech

vech Create the half-vectorization of a matrix

Description

vech Create the half-vectorization of a matrix

Usage

```
vech(x)
```

Arguments

x a matrix, the half-vectorization of which is desired

Details

This function returns the vectorized form of the lower triangle of a matrix, including the diagonal. The upper triangle is ignored with no checking that the provided matrix is symmetric.

Value

A vector containing the lower triangle of the matrix, including the diagonal.

Examples

```
vech(matrix(c(1, 0.5, 0.5, 1), nrow = 2, ncol = 2))
```

Index

* datasets

- hazard, 36
- inbreeding, 38
- potter, 56
- royal92, 66
- .adjBeta, 4
- .adjDirect, 5
- .adjIndexed, 6
- .assignParentValue, 7
- .computeTranspose, 8
- .loadOrComputeIsChild, 8
- .postProcessGedcom.legacy, 9

- addParentRow, 9
- addPersonToPed, 10
- addRowlessParents, 11
- allGens (calcAllGens), 15
- applyTagMappings, 12

- buildBetweenGenerations, 12
- buildWithinGenerations, 14

- calcAllGens, 15
- calcFamilySize, 16
- calcFamilySizeByGen, 16
- calculateCIs, 17
- calculateH, 18
- calculateRelatedness, 19
- checkIDs, 21
- checkIDUniqueness, 21
- checkParentIDs, 22
- checkParentSex, 23
- checkPedigreeNetwork, 24
- checkSex, 25, 66
- checkWithinRowDuplicates, 26
- collapseNames, 27
- com2links, 27, 28
- comp2vech, 28
- computeParentAdjacency, 29
- countPatternRows, 31

- createGenDataFrame, 31

- determineSex, 32
- dropLink, 33

- evenInsert (insertEven), 39

- famSizeCal (calcFamilySize), 16
- findBiggest, 33
- findbiggest (findBiggest), 33
- findOldest, 34
- findoldest (findOldest), 34
- fitComponentModel, 35

- gc, 28, 44, 46, 48, 50, 54

- hazard, 36

- identifyComponentModel, 37
- inbreeding, 38
- initializeRecord, 39
- insertEven, 39

- makeInbreeding, 40
- makeTwins, 41
- mapFAMS2parents, 42
- markPotentialChildren, 42

- parseNameLine, 43
- ped2add, 44
- ped2ce, 45
- ped2cn, 46
- ped2com, 5, 6, 30, 45, 47, 47, 49, 51–55
- ped2fam, 49
- ped2gen, 50
- ped2graph, 51
- ped2maternal, 52
- ped2mit, 53
- ped2mt (ped2mit), 53
- ped2paternal, 55
- postProcessGedcom, 56

potter, [56](#)
prepSummarizePedigrees, [57](#)
processEventLine, [58](#)
processParents, [59](#)

readGed (readGedcom), [59](#)
readGedcom, [59](#)
readgedcom (readGedcom), [59](#)
readWikifamilytree, [61](#)
recodeSex, [62](#)
related_coef (calculateRelatedness), [19](#)
repairIDs, [64](#)
repairParentIDs, [64](#)
repairSex, [65](#)
royal92, [66](#)

SimPed, [40](#)
SimPed (simulatePedigree), [67](#)
simulatePedigree, [67](#)
sizeAllGens (calcFamilySizeByGen), [16](#)
sliceFamilies, [69](#)
summariseFamilies (summarizeFamilies),
[70](#)
summarisefamilies (summarizeFamilies),
[70](#)
summariseMatrilines
 (summarizeMatrilines), [72](#)
summarisematrilines
 (summarizeMatrilines), [72](#)
summarisePatriline
 (summarizePatriline), [74](#)
summarisepatriline
 (summarizePatriline), [74](#)
summarisePedigrees
 (summarizePedigrees), [75](#)
summarisepedigrees
 (summarizePedigrees), [75](#)
summarizeFamilies, [70](#)
summarisefamilies (summarizeFamilies),
[70](#)
summarizeMatrilines, [72](#)
summarizematrilines
 (summarizeMatrilines), [72](#)
summarizePatriline, [74](#)
summarisepatriline
 (summarizePatriline), [74](#)
summarizePedigrees, [75](#)
summarisepedigrees
 (summarizePedigrees), [75](#)

traceTreePaths, [78](#)

validate_and_convert_matrix, [78](#)
vech, [79](#)