Package 'BGmisc'

June 16, 2024

Title An R Package for Extended Behavior Genetics Analysis

Version 1.3.1

Description The BGmisc R package offers a comprehensive suite of functions tailored for extended behavior genetics analysis, including model identification, calculating relatedness, pedigree conversion, pedigree simulation, and more.

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 igraph, kinship2, Matrix, stats, data.table, stringr

Suggests dplyr, EasyMx, knitr, OpenMx, rmarkdown, rticles, tidyverse, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Language en-US

NeedsCompilation no

Author S. Mason Garrison [aut, cre] (<https://orcid.org/0000-0002-4804-6003>), Michael D. Hunter [aut] (<https://orcid.org/0000-0002-3651-6709>), Xuanyu Lyu [aut] (<https://orcid.org/0000-0002-2841-5529>), Rachel N. Good [ctb], Jonathan D. Trattner [aut] (<https://orcid.org/0000-0002-1097-7603>, https://www.jdtrat.com/), S. Alexandra Burt [aut] (<https://orcid.org/0000-0001-5538-7431>)

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

Repository CRAN

Date/Publication 2024-06-16 18:00:02 UTC

Contents

`adjustKidsPerCouple` *Generate or Adjust Number of Kids per Couple Based on Mating Rate*

Description

This function generates or adjusts the number of kids per couple in a generation based on the specified average and whether the count should be randomly determined.

Usage

```
adjustKidsPerCouple(nMates, kpc, rd_kpc)
```

Arguments

`nMates`	Integer, the number of mated pairs in the generation.
`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.

Value

A numeric vector with the generated or adjusted number of kids per couple.

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

Description

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

Usage

```
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.

---

assignCoupleIds *Assign Couple IDs*

Description

This subfunction assigns a unique couple ID to each mated pair in the generation. Unmated individuals are assigned NA for their couple ID.

Usage

```
assignCoupleIds(df_Ngen)
```

Arguments

df_Ngen	The dataframe for the current generation, including columns for individual IDs and spouse IDs.

Value

The input dataframe augmented with a 'coupleId' column, where each mated pair has a unique identifier.

---

buildBetweenGenerations *Process Generation Connections*

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,
  marR,
  sexR,
  kpc,
  rd_kpc
)
```

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, print 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, 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.

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)
```

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.

Value

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

calculateH *Falconer's Formula*

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 *Calculate Relatedness Coefficient*

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,
  ...
)
```

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*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 *Validates and Optionally Repairs Unique IDs in a Pedigree Dataframe*

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)
```

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

Description

This function performs two main tasks: 1. Optionally recodes the 'sex' variable based on given codes for males and females. 2. Optionally repairs the sex coding based on specified logic, facilitating the accurate construction of genetic pedigrees.

Usage

```
checkSex(
  ped,
  code_male = NULL,
  code_female = NULL,
  verbose = FALSE,
  repair = FALSE
)
```

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.
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.

Details

This function uses the terms 'male' and 'female' in a biological context, based on chromosomes and other biologically-based characteristics relevant to genetic studies. This usage is not intended to negate the personal gender identity of any individual.

We recognize the importance of using language and methodologies that affirm and respect all gender identities. While this function focuses on chromosomal information necessary for constructing genetic pedigrees, we affirm that gender is a spectrum, encompassing a wide range of identities beyond the binary. The developers of this package express unequivocal support for folx in the transgender and LGBTQ+ communities. We respect the complexity of gender identity and acknowledge the distinction between the biological aspect of sex used for genetic analysis (genotype) and the broader, richer concept of gender identity (phenotype).

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)
```

---

comp2vech *comp2vech 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)))
```

createGenDataFrame *Create Data Frame for Generation*

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 *Determine Sex of Offspring*

Description

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

Usage

```
determineSex(idGen, sexR)
```

Arguments

idGen	Vector of IDs for the generation.
sexR	Numeric value indicating the sex ratio (proportion of males).

Value

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

`dropLink` *dropLink 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`*.*

Description

dropLink 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.

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

Description

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

Usage

```
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.

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

Description

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

Usage

```
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.

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

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")
selVars <- 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 *Simulated pedigree with two extended families and an age-related hazard*

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

...	Comma-separated relatedness component matrices representing the variance components of the model.
verbose	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.

Usage

```
data(inbreeding)
```

Format

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

Details

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.

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

`inferRelatedness` *Infer Relatedness Coefficient*

Description

This function infers the relatedness coefficient between two groups based on the observed correlation between their additive genetic variance and shared environmental variance.

Usage

```
inferRelatedness(obsR, aceA = 0.9, aceC = 0, sharedC = 0)
```

Arguments

`obsR`	Numeric. Observed correlation between the two groups. Must be between -1 and 1.
`aceA`	Numeric. Proportion of variance attributable to additive genetic variance. Must be between 0 and 1. Default is 0.9.
`aceC`	Numeric. Proportion of variance attributable to shared environmental variance. Must be between 0 and 1. Default is 0.
`sharedC`	Numeric. Proportion of shared environment shared between the two individuals. Must be between 0 and 1. Default is 0.

Details

The function uses the ACE (Additive genetic, Common environmental, and Unique environmental) model to infer the relatedness between two individuals or groups. By considering the observed correlation (‘obsR‘), the proportion of variance attributable to additive genetic variance (‘aceA‘), and the proportion of shared environmental variance (‘aceC‘), it calculates the relatedness coefficient.

Value

Numeric. The calculated relatedness coefficient (‘est_r‘).

Examples

```
## Not run:
# Infer the relatedness coefficient:
inferRelatedness(obsR = 0.5, aceA = 0.9, aceC = 0, sharedC = 0)

## End(Not run)
```

makeInbreeding *makeInbreeding 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`.

Description

makeInbreeding 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

`ped`	A `data.frame` in the same format as the output of `simulatePedigree`.
`ID_mate1`	A vector of `ID` of the first mate. If not provided, the function will randomly select two individuals from the second generation.
`ID_mate2`	A vector of `ID` of the second mate.
`verbose`	logical. If TRUE, print progress through stages of algorithm
`gen_inbred`	A vector of `generation` of the twin to be imputed.
`type_inbred`	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.

makeTwins *makeTwins 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*.*

Description

makeTwins 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
)
```

Arguments

ped	A data.frame in the same format as the output of simulatePedigree.
ID_twin1	A vector of ID of the first twin.
ID_twin2	A vector of ID of the second twin.
gen_twin	A vector of generation of the twin to be imputed.
verbose	logical. If TRUE, print progress through stages of algorithm

Value

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

markPotentialChildren *Mark and Assign children*

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)
```

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.

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‘).

---

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 = FALSE,
  verbose = FALSE,
  gc = FALSE,
  flatten.diag = FALSE,
  standardize.colnames = TRUE,
  tcross.alt.crossprod = FALSE,
  tcross.alt.star = FALSE
)
```

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
tcross.alt.crossprod	logical. If TRUE, use alternative method of using Crossprod function for computing the transpose
tcross.alt.star	logical. If TRUE, use alternative method of using %*% for computing the transpose

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

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

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

Description

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

Usage

```
ped2cn(
  ped,
  max.gen = 25,
  sparse = FALSE,
  verbose = FALSE,
  gc = FALSE,
  flatten.diag = FALSE,
  standardize.colnames = TRUE,
  tcross.alt.crossprod = FALSE,
  tcross.alt.star = FALSE
)
```

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
tcross.alt.crossprod	logical. If TRUE, use alternative method of using Crossprod function for computing the transpose
tcross.alt.star	logical. If TRUE, use alternative method of using %*% for computing the transpose

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 = FALSE,
  verbose = FALSE,
  gc = FALSE,
  flatten.diag = FALSE,
  standardize.colnames = TRUE,
  tcross.alt.crossprod = FALSE,
  tcross.alt.star = FALSE,
  ...
)
```

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
tcross.alt.crossprod	logical. If TRUE, use alternative method of using Crossprod function for computing the transpose
tcross.alt.star	logical. If TRUE, use alternative method of using %*% for computing the transpose
...	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

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

ped2graph *Turn a pedigree into a graph*

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

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

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 = FALSE,
  verbose = FALSE,
  gc = FALSE,
  flatten.diag = FALSE,
  standardize.colnames = TRUE,
  tcross.alt.crossprod = FALSE,
  tcross.alt.star = FALSE
)
```

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
`tcross.alt.crossprod`	logical. If TRUE, use alternative method of using Crossprod function for computing the transpose
`tcross.alt.star`	logical. If TRUE, use alternative method of using %*% for computing the transpose

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` *Add a paternal line ID variable to a pedigree*

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

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

plotPedigree *plotPedigree A wrapped function to plot simulated pedigree from function* `simulatePedigree`. *This function require the installation of package* `kinship2`.

Description

plotPedigree A wrapped function to plot simulated pedigree from function `simulatePedigree`. This function require the installation of package `kinship2`.

Usage

```
plotPedigree(
  ped,
  code_male = NULL,
  verbose = FALSE,
  affected = NULL,
  cex = 0.5,
  col = 1,
  symbolsize = 1,
  branch = 0.6,
  packed = TRUE,
  align = c(1.5, 2),
  width = 8,
  density = c(-1, 35, 65, 20),
  mar = c(2.1, 1, 2.1, 1),
  angle = c(90, 65, 40, 0),
  keep.par = FALSE,
  pconnect = 0.5,
  ...
)
```

Arguments

`ped`	The simulated pedigree data.frame from function `simulatePedigree`. Or a pedigree dataframe with the same colnames as the dataframe simulated from function `simulatePedigree`.
`code_male`	This optional input allows you to indicate what value in the sex variable codes for male. Will be recoded as "M" (Male). If `NULL`, no recoding is performed.
`verbose`	logical If TRUE, prints additional information. Default is FALSE.
`affected`	This optional parameter can either be a string specifying the column name that indicates affected status or a numeric/logical vector of the same length as the number of rows in 'ped'. If `NULL`, no affected status is assigned.
`cex`	The font size of the IDs for each individual in the plot.
`col`	color for each id. Default assigns the same color to everyone.

symbolsize	controls symbolsize. Default=1.
branch	defines how much angle is used to connect various levels of nuclear families.
packed	default=T. If T, uniform distance between all individuals at a given level.
align	these parameters control the extra effort spent trying to align children underneath parents, but without making the pedigree too wide. Set to F to speed up plotting.
width	default=8. For a packed pedigree, the minimum width allowed in the realignment of pedigrees.
density	defines density used in the symbols. Takes up to 4 different values.
mar	margin parmeters, as in the par function
angle	defines angle used in the symbols. Takes up to 4 different values.
keep.par	Default = F, allows user to keep the parameter settings the same as they were for plotting (useful for adding extras to the plot)
pconnect	when connecting parent to children the program will try to make the connecting line as close to vertical as possible, subject to it lying inside the endpoints of the line that connects the children by at least pconnect people. Setting this option to a large number will force the line to connect at the midpoint of the children.
...	Extra options that feed into the plot function.

Value

A plot of the provided pedigree

---

potter *Fictional pedigree data on a wizarding family*

Description

A dataset created purely from imagination that includes a subset of the Potter extended family.

Usage

```
data(potter)
```

Format

A data frame (and ped object) with 36 rows and 8 variables

Details

The variables are as follows:

- personID: Person identification variable
- famID: Family identification variable
- name: 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

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

---

readGedcom *Read a GEDCOM File*

Description

This function reads a GEDCOM file and parses it into a structured data frame of individuals. Inspired by https://raw.githubusercontent.com/jjfitz/readgedcom/master/R/read_gedcom.R

Usage

```
readGedcom(
  file_path,
  verbose = FALSE,
  add_parents = TRUE,
  remove_empty_cols = TRUE,
  combine_cols = TRUE,
  skinny = FALSE
)
```

Arguments

file_path	The path to the GEDCOM file.
verbose	A logical value indicating whether to print messages.
add_parents	A logical value indicating whether to add parents to the data frame.
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.
skinny	A logical value indicating whether to return a skinny data frame.

Value

A data frame containing information about individuals, with the following potential columns: - ‘id‘: ID of the individual - ‘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

---

`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,
  recode_male = "M",
  recode_female = "F",
  recode_na = NA_character_
)
```

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_na`	The current value used for missing values.
`code_female`	The current code used to represent females in the ’sex’ column. If both are NULL, no recoding is performed.
`recode_male`	The value to use for males. Default is "M"
`recode_female`	The value to use for females. Default is "F"
`recode_na`	The value to use for missing values. Default is NA_character_

Details

This function uses the terms 'male' and 'female' in a biological context, based on chromosomes and other biologically-based characteristics relevant to genetic studies. This usage is not intended to negate the personal gender identity of any individual.

We recognize the importance of using language and methodologies that affirm and respect all gender identities. While this function focuses on chromosomal information necessary for constructing genetic pedigrees, we affirm that gender is a spectrum, encompassing a wide range of identities beyond the binary. The developers of this package express unequivocal support for folx in the transgender and LGBTQ+ communities. We respect the complexity of gender identity and acknowledge the distinction between the biological aspect of sex used for genetic analysis (genotype) and the broader, richer concept of gender identity (phenotype).

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

`plotPedigree`

`relatedness` *relatedness (Deprecated)*

Description

When calling this function, a warning will be issued about its deprecation.

Usage

```
relatedness(...)
```

Arguments

`...`	Arguments to be passed to 'inferRelatedness'.

Details

This function is a wrapper around the new 'inferRelatedness' function. 'relatedness' has been deprecated, and it's advised to use 'inferRelatedness' directly.

Value

The same result as calling 'inferRelatedness'.

See Also

inferRelatedness for the updated function.

Examples

```
## Not run:
# This is an example of the deprecated function:
relatedness(...)
# It is recommended to use:
inferRelatedness(...)

## End(Not run)
```

---

related_coef *related_coef (Deprecated)*

Description

When calling this function, a warning will be issued about its deprecation.

Usage

```
related_coef(...)
```

Arguments

...	Arguments to be passed to 'calculateRelatedness'.

Details

This function is a wrapper around the new 'calculateRelatedness' function. 'related_coef' has been deprecated, and it's advised to use 'calculateRelatedness' directly.

Value

The same result as calling 'calculateRelatedness'.

See Also

calculateRelatedness for the updated function.

Examples

```
## Not run:
# This is an example of the deprecated function:
related_coef(...)
# It is recommended to use:
calculateRelatedness(...)

## End(Not run)
```

repairIDs *Repair Missing IDs*

Description

This function repairs missing IDs in a pedigree.

Usage

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

Arguments

ped	A pedigree object
verbose	A logical indicating whether to print progress messages

Value

A corrected pedigree

repairSex *Repairs Sex Coding in a Pedigree Dataframe*

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)
```

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.

Details

This function uses the terms 'male' and 'female' in a biological context, based on chromosomes and other biologically-based characteristics relevant to genetic studies. This usage is not intended to negate the personal gender identity of any individual.

We recognize the importance of using language and methodologies that affirm and respect all gender identities. While this function focuses on chromosomal information necessary for constructing genetic pedigrees, we affirm that gender is a spectrum, encompassing a wide range of identities beyond the binary. The developers of this package express unequivocal support for folx in the transgender and LGBTQ+ communities. We respect the complexity of gender identity and acknowledge the distinction between the biological aspect of sex used for genetic analysis (genotype) and the broader, richer concept of gender identity (phenotype).

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)
```

---

resample *Resample Elements of a Vector*

Description

This function performs resampling of the elements in a vector ‘x‘. It randomly shuffles the elements of ‘x‘ and returns a vector of the resampled elements. If ‘x‘ is empty, it returns ‘NA_integer_‘.

Usage

```
resample(x, ...)
```

Arguments

`x`	A vector containing the elements to be resampled. If ‘x‘ is empty, the function will return ‘NA_integer_‘.
`...`	Additional arguments passed to ‘sample.int‘, such as ‘size‘ for the number of items to sample and ‘replace‘ indicating whether sampling should be with replacement.

Value

A vector of resampled elements from ‘x‘. If ‘x‘ is empty, returns ‘NA_integer_‘. The length and type of the returned vector depend on the input vector ‘x‘ and the additional arguments provided via ‘...‘.

`SimPed` *SimPed (Deprecated)*

Description

When calling this function, a warning will be issued about its deprecation.

Usage

```
SimPed(...)
```

Arguments

`...`	Arguments to be passed to ‘simulatePedigree‘.

Details

This function is a wrapper around the new ‘simulatePedigree‘ function. ‘SimPed‘ has been deprecated, and it’s advised to use ‘simulatePedigree‘ directly.

Value

The same result as calling ‘simulatePedigree‘.

See Also

`simulatePedigree` for the updated function.

Examples

```
## Not run:
# This is an example of the deprecated function:
SimPed(...)
# It is recommended to use:
simulatePedigree(...)

## End(Not run)
```

simulatePedigree	*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.*

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
)
```

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, print progress through stages of algorithm

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 his/her 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.

`sizeAllGens` *sizeAllGens An internal supporting function for* `simulatePedigree`.

Description

sizeAllGens An internal supporting function for `simulatePedigree`.

Usage

```
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.

`summarizeFamilies` *Summarize the families in a pedigree*

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,
  nbiggest = 5,
  noldest = 5,
  skip_var = NULL,
  five_num_summary = FALSE,
  verbose = 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`	Optional column name for birth year.
`founder_sort_var`	The variable to sort the founders by. If NULL, the founders will be sorted by birth year (‘byr‘) if that’s present and by ‘personID‘ otherwise.
`include_founder`	Logical, if TRUE, include the founder of each line in the summary statistics.
`nbiggest`	The number of biggest lines to return.
`noldest`	The number of oldest lines to return.

skip_var A character vector of variables to skip when calculating summary statistics.

five_num_summary Logical, if TRUE, include the 5-number summary (min, Q1, median, Q3, max) in the summary statistics.

verbose Logical, if TRUE, print progress messages.

See Also

[summarizePedigrees ()]

`summarizeMatrilines` *Summarize the maternal lines in a pedigree*

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,
  nbiggest = 5,
  noldest = 5,
  skip_var = NULL,
  five_num_summary = FALSE,
  verbose = 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	Optional column name for birth year.
include_founder	Logical, if TRUE, include the founder of each line in the summary statistics.
founder_sort_var	The variable to sort the founders by. If NULL, the founders will be sorted by birth year (‘byr‘) if that’s present and by ‘personID‘ otherwise.
nbiggest	The number of biggest lines to return.
noldest	The number of oldest lines to return.
skip_var	A character vector of variables to skip when calculating summary statistics.
five_num_summary	Logical, if TRUE, include the 5-number summary (min, Q1, median, Q3, max) in the summary statistics.
verbose	Logical, if TRUE, print progress messages.

See Also

[summarizePedigrees ()]

`summarizePatrilines` *Summarize the paternal lines in a pedigree*

Description

Summarize the paternal lines in a pedigree

Usage

```
summarizePatrilines(
  ped,
  famID = "famID",
  personID = "ID",
  momID = "momID",
  dadID = "dadID",
  matID = "matID",
  patID = "patID",
  byr = NULL,
  founder_sort_var = NULL,
  include_founder = FALSE,
  nbiggest = 5,
  noldest = 5,
  skip_var = NULL,
  five_num_summary = FALSE,
  verbose = 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	Optional column name for birth year.
founder_sort_var	The variable to sort the founders by. If NULL, the founders will be sorted by birth year (‘byr‘) if that’s present and by ‘personID‘ otherwise.
include_founder	Logical, if TRUE, include the founder of each line in the summary statistics.
nbiggest	The number of biggest lines to return.
noldest	The number of oldest lines to return.
skip_var	A character vector of variables to skip when calculating summary statistics.
five_num_summary	Logical, if TRUE, include the 5-number summary (min, Q1, median, Q3, max) in the summary statistics.
verbose	Logical, if TRUE, print progress messages.

See Also

[summarizePedigrees ()]

---

summarizePedigrees *Summarize Pedigree Data*

Description

This function summarizes pedigree data, including calculating summary statistics for all numeric variables, and finding the originating member for each family, maternal, and paternal line.

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,
  nbiggest = 5,
  noldest = 5,
  skip_var = NULL,
  five_num_summary = FALSE,
  verbose = 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
type	The type of summary statistics to calculate. Options are "fathers", "mothers", and "families".
byr	Optional column name for birth year.
include_founder	Logical, if TRUE, include the founder of each line in the summary statistics.
founder_sort_var	The variable to sort the founders by. If NULL, the founders will be sorted by birth year (‘byr‘) if that’s present and by ‘personID‘ otherwise.
nbiggest	The number of biggest lines to return.
noldest	The number of oldest lines to return.
skip_var	A character vector of variables to skip when calculating summary statistics.
five_num_summary	Logical, if TRUE, include the 5-number summary (min, Q1, median, Q3, max) in the summary statistics.
verbose	Logical, if TRUE, print progress messages.

Value

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

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