Package 'simfam'

October 6, 2021

Title Simulate and Model Family Pedigrees with Structured Founders

Version 1.0.3

Description
The focus is on simulating and modeling families with founders drawn from a structured population (for example, with different ancestries or other potentially non-family relatedness), in contrast to traditional pedigree analysis that treats all founders as equally unrelated. Main function simulates a random pedigree for many generations, avoiding close relatives, pairing closest individuals according to a 1D geography and their randomly-drawn sex, and with variable children sizes to result in a target population size per generation. Auxiliary functions calculate kinship matrices, admixture matrices, and draw random genotypes across arbitrary pedigree structures starting from the corresponding founder values. The code is built around the plink FAM table format for pedigrees. Partially described in Yao and Ochoa (2019) <doi:10.1101/858399>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.1.2

Imports Rcpp, stats, tibble

Suggests testthat (>= 3.0.0), popkin, bnpsd (>= 1.3.2), kinship2, RColorBrewer, knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

LinkingTo Rcpp

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

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

NeedsCompilation yes

Author Alejandro Ochoa [aut, cre] (<https://orcid.org/0000-0003-4928-3403>)

Maintainer Alejandro Ochoa <alejandro.ochoa@duke.edu>

Repository CRAN

Date/Publication 2021-10-05 23:20:10 UTC

R topics documented:

`admix_fam` *Calculate admixture matrix of a pedigree with known admixture of founders*

Description

Calculates a full admixture proportions matrix (for all individuals in the provided pedigree FAM table) starting from the admixture proportions of the founders as provided.

Usage

```
admix_fam(admix, fam, missing_vals = c("", 0))
```

Arguments

`admix`	The admixture proportions matrix of the founders (individuals along rows and ancestries along columns). This matrix must have row names that identify each founder (matching codes in `fam$id`). Individuals may be in a different order than `fam$id`. Extra individuals in `admix` but absent in `fam$id` will be silently ignored. All values should be non-negative and each row of `admix` should sum to one; for speed, this code does not check that `admix` is valid, just averages data as-is.
`fam`	The pedigree data.frame, in plink FAM format. Only columns `id`, `pat`, and `mat` are required. `id` must be unique and non-missing. Founders must be present, and their `pat` and `mat` values must be missing (see below). Non-founders must have both their parents be non-missing. Parents must appear earlier than their children in the table.
`missing_vals`	The list of ID values treated as missing. `NA` is always treated as missing. By default, the empty string ('') and zero (0) are also treated as missing (remove values from here if this is a problem).

Value

The admixture proportions matrix of the entire `fam` table, based on the admixture of the founders. These are expectations, calculated for each individual as the average ancestry proportion of the parents. The rows of this admixture matrix correspond to `fam$id` in that order. The columns (ancestries) are the same as in the input `admix`.

See Also

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

Examples

```
# The smallest pedigree, two parents and a child.
# A minimal fam table with the three required columns.
# Note "mother" and "father" have missing parent IDs, while "child" does not
library(tibble)
fam <- tibble(
  id = c('father', 'mother', 'child'),
  pat = c(NA, NA, 'father'),
  mat = c(NA, NA, 'mother')
)

# admixture proportions of the parents
admix <- rbind( c(0.3, 0.3, 0.4), c(0.5, 0.25, 0.25) )
# Name the parents with same codes as in `fam`
# (order can be different)
rownames( admix ) <- c('mother', 'father')
# name ancestries too
colnames( admix ) <- c('African', 'European', 'Asian')

# Calculate the full admixture proportions matrix
admix_all <- admix_fam( admix, fam )

# This is a 3x3 matrix with row names matching fam$id.
# The parent submatrix equals the input (reordered),
# but now there's admixture to the child too (averages of parents)
admix_all
```

`admix_last_gen` *Calculate admixture matrix for last generation of a pedigree with admixture of founders*

Description

A wrapper around the more general `admix_fam()`, specialized to save memory when only the last generation is desired (`admix_fam()` returns admixture for the entire pedigree in a single matrix). This function assumes that generations are non-overlapping (met by the output of `sim_pedigree()`), in which case each generation `g` can be drawn from generation `g-1` data only. That way, only two

consecutive generations need be in memory at any given time. The partitioning of individuals into generations is given by the `ids` parameter (again matches the output of `sim_pedigree()`).

Usage

```
admix_last_gen(admix, fam, ids, missing_vals = c("", 0))
```

Arguments

`admix`	The admixture proportions matrix of the founders (individuals along rows and ancestries along columns). This matrix must have row names that identify each founder (matching codes in `fam$id`). Individuals may be in a different order than `fam$id`. Extra individuals in `admix` but absent in `fam$id` will be silently ignored. All values should be non-negative and each row of `admix` should sum to one; for speed, this code does not check that `admix` is valid, just averages data as-is.
`fam`	The pedigree data.frame, in plink FAM format. Only columns `id`, `pat`, and `mat` are required. `id` must be unique and non-missing. Founders must be present, and their `pat` and `mat` values must be missing (see below). Non-founders must have both their parents be non-missing. Parents must appear earlier than their children in the table.
`ids`	A list containing vectors of IDs for each generation. All these IDs must be present in `fam$id`. If IDs in `fam` and `ids` do not fully agree, the code processes the IDs in the intersection, which is helpful when `fam` is pruned but `ids` is the original (larger) set.
`missing_vals`	The list of ID values treated as missing. `NA` is always treated as missing. By default, the empty string ('') and zero (0) are also treated as missing (remove values from here if this is a problem).

Value

The admixture proportions matrix of the last generation (the intersection of `ids[ length(ids) ]` and `fam$id`). The rows of this matrix are last-generation individuals in the order that they appear in `fam$id`.

See Also

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

Examples

```
# A small pedigree, two parents and two children.
# A minimal fam table with the three required columns.
# Note "mother" and "father" have missing parent IDs, while children do not
library(tibble)
fam <- tibble(
  id = c('father', 'mother', 'child', 'sib'),
  pat = c(NA, NA, 'father', 'father'),
  mat = c(NA, NA, 'mother', 'mother')
```

```
)
# need an `ids` list separating the generations
ids <- list( c('father', 'mother'), c('child', 'sib') )

# admixture proportions of the parents
admix <- rbind( c(0.3, 0.3, 0.4), c(0.5, 0.25, 0.25) )
# Name the parents with same codes as in `fam`
# (order can be different)
rownames( admix ) <- c('mother', 'father')
# name ancestries too
colnames( admix ) <- c('African', 'European', 'Asian')

# calculate the admixture matrix of the children
admix2 <- admix_last_gen( admix, fam, ids )
admix2
```

`draw_sex` *Draw sex values randomly for a list of individuals*

Description

Each individual has their sex drawn between male and female with equal probability. Sex is encoded numerically following the convention for plink FAM files (see below).

Usage

```
draw_sex(n)
```

Arguments

`n`	The number of individuals.

Value

The length-`n` vector of integer sex assignments: `1L` corresponds to male, `2L` to female.

See Also

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

Examples

```
draw_sex( 10 )
```

geno_fam *Draw random genotypes on a pedigree with known founder genotypes*

Description

Constructs a random genotype matrix (for all individuals in the provided pedigree FAM table) starting from the genotype matrix of the founders as provided.

Usage

```
geno_fam(X, fam, missing_vals = c("", 0))
```

Arguments

X	The genotype matrix of the founders (loci along rows, individuals along columns). This matrix must have column names that identify each founder (matching codes in fam$id). Individuals may be in a different order than fam$id. Extra individuals in admix but absent in fam$id will be silently ignored. All values should be in c(0L,1L,2L); for speed, this code does not check that X is valid (i.e. fractional values between 0 and 2 may not cause errors).
fam	The pedigree data.frame, in plink FAM format. Only columns id, pat, and mat are required. id must be unique and non-missing. Founders must be present, and their pat and mat values must be missing (see below). Non-founders must have both their parents be non-missing. Parents must appear earlier than their children in the table.
missing_vals	The list of ID values treated as missing. NA is always treated as missing. By default, the empty string ('') and zero (0) are also treated as missing (remove values from here if this is a problem).

Value

The random genotype matrix of the entire fam table, starting from the genotypes of the founders. The columns of this matrix correspond to fam$id in that order. The rows (loci) are the same as in the input X.

See Also

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

Examples

```
# The smallest pedigree, two parents and a child.
# A minimal fam table with the three required columns.
# Note "mother" and "father" have missing parent IDs, while "child" does not
library(tibble)
fam <- tibble(
  id = c('father', 'mother', 'child'),
```

```
  pat = c(NA, NA, 'father'),
  mat = c(NA, NA, 'mother')
)

# genotypes of the parents at 4 loci
X <- cbind( c(1, 2, 0, 2), c(0, 2, 2, 1) )
# Name the parents with same codes as in `fam`
# (order can be different)
colnames( X ) <- c('mother', 'father')
# name loci too
rownames( X ) <- paste0( 'rs', 1:4 )

# Draw the full genotype matrix
X_all <- geno_fam( X, fam )

# This is a 4x3 matrix with column names matching fam$id.
# The parent submatrix equals the input (reordered),
# but now there's random genotypes for the child too
X_all
```

`geno_last_gen` *Draw random genotypes for last generation of a pedigree with known founder genotypes*

Description

A wrapper around the more general `geno_fam()`, specialized to save memory when only the last generation is desired (`geno_fam()` returns genotypes for the entire pedigree in a single matrix). This function assumes that generations are non-overlapping (met by the output of `sim_pedigree()`), in which case each generation `g` can be drawn from generation `g-1` data only. That way, only two consecutive generations need be in memory at any given time. The partitioning of individuals into generations is given by the `ids` parameter (again matches the output of `sim_pedigree()`).

Usage

```
geno_last_gen(X, fam, ids, missing_vals = c("", 0))
```

Arguments

`X`	The genotype matrix of the founders (loci along rows, individuals along columns). This matrix must have column names that identify each founder (matching codes in `fam$id`). Individuals may be in a different order than `fam$id`. Extra individuals in `admix` but absent in `fam$id` will be silently ignored. All values should be in `c(0L,1L,2L)`; for speed, this code does not check that `X` is valid (i.e. fractional values between 0 and 2 may not cause errors).
`fam`	The pedigree data.frame, in plink FAM format. Only columns `id`, `pat`, and `mat` are required. `id` must be unique and non-missing. Founders must be present,

and their pat and mat values must be missing (see below). Non-founders must have both their parents be non-missing. Parents must appear earlier than their children in the table.

ids — A list containing vectors of IDs for each generation. All these IDs must be present in fam$id. If IDs in fam and ids do not fully agree, the code processes the IDs in the intersection, which is helpful when fam is pruned but ids is the original (larger) set.

missing_vals — The list of ID values treated as missing. NA is always treated as missing. By default, the empty string ('') and zero (0) are also treated as missing (remove values from here if this is a problem).

Value

The random genotype matrix of the last generation (the intersection of ids[length(ids)] and fam$id). The columns of this matrix are last-generation individuals in the order that they appear in fam$id. The rows (loci) are the same as in the input X.

See Also

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

Examples

```
# A small pedigree, two parents and two children.
# A minimal fam table with the three required columns.
# Note "mother" and "father" have missing parent IDs, while children do not
library(tibble)
fam <- tibble(
  id = c('father', 'mother', 'child', 'sib'),
  pat = c(NA, NA, 'father', 'father'),
  mat = c(NA, NA, 'mother', 'mother')
)
# need an `ids` list separating the generations
ids <- list( c('father', 'mother'), c('child', 'sib') )

# genotypes of the parents at 4 loci
X <- cbind( c(1, 2, 0, 2), c(0, 2, 2, 1) )
# Name the parents with same codes as in `fam`
# (order can be different)
colnames( X ) <- c('mother', 'father')
# name loci too
rownames( X ) <- paste0( 'rs', 1:4 )

# Draw the genotype matrix of the children
X2 <- geno_last_gen( X, fam, ids )
X2
```

`kinship_fam` *Calculate kinship matrix of a pedigree with structured founders*

Description

Calculates a full kinship matrix (between all individuals in the provided pedigree FAM table) taking into account the relatedness of the founders as provided. Output agrees with `kinship2::kinship()` but only when founders are unrelated/outbred (in other words, that function does not allow relatedness between founders).

Usage

```
kinship_fam(kinship, fam, missing_vals = c("", 0))
```

Arguments

`kinship`	The kinship matrix of the founders. This matrix must have column and row names that identify each founder (matching codes in `fam$id`). Individuals may be in a different order than `fam$id`. Extra individuals in `kinship` but absent in `fam$id` will be silently ignored. A traditional pedigree calculation would use `kinship = diag(n)/2` (plus appropriate column/row names), where `n` is the number of founders, to model unrelated and outbred founders. However, if `kinship` measures the population kinship estimates between founders, the output is also a population kinship matrix (which combines the structural/ancestral and local/pedigree relatedness values into one).
`fam`	The pedigree data.frame, in plink FAM format. Only columns `id`, `pat`, and `mat` are required. `id` must be unique and non-missing. Founders must be present, and their `pat` and `mat` values must be missing (see below). Non-founders must have both their parents be non-missing. Parents must appear earlier than their children in the table.
`missing_vals`	The list of ID values treated as missing. `NA` is always treated as missing. By default, the empty string (") and zero (0) are also treated as missing (remove values from here if this is a problem).

Value

The kinship matrix of the entire `fam` table, taking the relatedness of the founders into account. The rows and columns of this kinship matrix correspond to `fam$id` in that order.

See Also

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

Examples

```
# The smallest pedigree, two parents and a child.
# A minimal fam table with the three required columns.
# Note "mother" and "father" have missing parent IDs, while "child" does not
library(tibble)
fam <- tibble(
  id = c('father', 'mother', 'child'),
  pat = c(NA, NA, 'father'),
  mat = c(NA, NA, 'mother')
)

# Kinship of the parents, here two unrelated/outbred individuals:
kinship <- diag(2)/2
# Name the parents with same codes as in `fam`
# (order can be different)
colnames( kinship ) <- c('mother', 'father')
rownames( kinship ) <- c('mother', 'father')
# For a clearer example, make the father slightly inbred
# (a self-kinship value that exceeds 1/2):
kinship[2,2] <- 0.6

# Calculate the full kinship matrix
kinship_all <- kinship_fam( kinship, fam )

# This is a 3x3 matrix with row/col names matching fam$id.
# The parent submatrix equals the input (reordered),
# but now there's relatedness to the child too
kinship_all
```

`kinship_last_gen` *Calculate kinship matrix for last generation of a pedigree with structured founders*

Description

A wrapper around the more general `kinship_fam()`, specialized to save memory when only the last generation is desired (`kinship_fam()` returns kinship for the entire pedigree in a single matrix). This function assumes that generations are non-overlapping (met by the output of `sim_pedigree()`), in which case each generation `g` can be drawn from generation `g-1` data only. That way, only two consecutive generations need be in memory at any given time. The partitioning of individuals into generations is given by the `ids` parameter (again matches the output of `sim_pedigree()`).

Usage

```
kinship_last_gen(kinship, fam, ids, missing_vals = c("", 0))
```

Arguments

kinship	The kinship matrix of the founders. This matrix must have column and row names that identify each founder (matching codes in fam$id). Individuals may be in a different order than fam$id. Extra individuals in kinship but absent in fam$id will be silently ignored. A traditional pedigree calculation would use kinship = diag(n)/2 (plus appropriate column/row names), where n is the number of founders, to model unrelated and outbred founders. However, if kinship measures the population kinship estimates between founders, the output is also a population kinship matrix (which combines the structural/ancestral and local/pedigree relatedness values into one).
fam	The pedigree data.frame, in plink FAM format. Only columns id, pat, and mat are required. id must be unique and non-missing. Founders must be present, and their pat and mat values must be missing (see below). Non-founders must have both their parents be non-missing. Parents must appear earlier than their children in the table.
ids	A list containing vectors of IDs for each generation. All these IDs must be present in fam$id. If IDs in fam and ids do not fully agree, the code processes the IDs in the intersection, which is helpful when fam is pruned but ids is the original (larger) set.
missing_vals	The list of ID values treated as missing. NA is always treated as missing. By default, the empty string ('') and zero (0) are also treated as missing (remove values from here if this is a problem).

Value

The kinship matrix of the last generation (the intersection of ids[length(ids)] and fam$id). The columns/rows of this matrix are last-generation individuals in the order that they appear in fam$id.

See Also

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

Examples

```
# A small pedigree, two parents and two children.
# A minimal fam table with the three required columns.
# Note "mother" and "father" have missing parent IDs, while children do not
library(tibble)
fam <- tibble(
  id = c('father', 'mother', 'child', 'sib'),
  pat = c(NA, NA, 'father', 'father'),
  mat = c(NA, NA, 'mother', 'mother')
)
# need an `ids` list separating the generations
ids <- list( c('father', 'mother'), c('child', 'sib') )

# Kinship of the parents, here two unrelated/outbred individuals:
kinship <- diag(2)/2
```

```
# Name the parents with same codes as in `fam`
# (order can be different)
colnames( kinship ) <- c('mother', 'father')
rownames( kinship ) <- c('mother', 'father')
# For a clearer example, make the father slightly inbred
# (a self-kinship value that exceeds 1/2):
kinship[2,2] <- 0.6

# calculate the kinship matrix of the children
kinship2 <- kinship_last_gen( kinship, fam, ids )
kinship2
```

---

prune_fam *Remove non-ancestors of a set of individuals from pedigree*

Description

This function accepts an input pedigree and a list of individuals of interest, and returns the subset of the pedigree including only the individuals of interest and their direct ancestors. This is useful in simulations, to avoid modeling/drawing genotypes of individuals without descendants in the last generation.

Usage

```
prune_fam(fam, ids, missing_vals = c("", 0))
```

Arguments

fam	The pedigree data.frame, in plink FAM format. Only columns id, pat, and mat are required. id must be unique and non-missing. Founders must be present, and their pat and mat values must be missing (see below). Non-founders must have both their parents be non-missing. Parents must appear earlier than their children in the table.
ids	The list of individuals of interest, whose ancestors we want to keep. All must be present in fam$id.
missing_vals	The list of ID values treated as missing. NA is always treated as missing. By default, the empty string (") and zero (0) are also treated as missing (remove values from here if this is a problem).

Value

The filtered FAM table with non-ancestors of ids excluded. IDs that are NA-equivalent (see missing_vals) will be mapped to NA.

Examples

```
# construct a family with three founders, but one "bob" has no descendants
library(tibble)
fam <- tibble(
    id  = c('mom', 'dad', 'bob', 'child'),
    pat = c(   NA,    NA,    NA,   'dad'),
    mat = c(   NA,    NA,    NA,   'mom')
)
# only want 'child' and its ancestors
ids <- 'child'
fam2 <- prune_fam( fam, ids )
# the filtered pedigree has "bob" removed:
fam2
```

`sim_pedigree` *Construct a random pedigree*

Description

Specify the number of individuals per generation, and some other optional parameters, and a single pedigree with those properties will be simulated, where close relatives are never paired, sex is drawn randomly per individual and pairings are strictly across opposite-sex individuals, and otherwise closest individuals (on an underlying 1D geography given by their index) are paired in a random order. Pairs are reordered based on the average of their indexes, where their children are placed (determines their indexes in the 1D geography). The procedure may leave some individuals unpaired in the next generation, and family sizes vary randomly (with a fixed minimum family size) to achieve the desired population size in each generation.

Usage

```
sim_pedigree(
  n,
  G = length(n),
  sex = draw_sex(n[1]),
  kinship_local = diag(n[1])/2,
  cutoff = 1/4^3,
  children_min = 1L,
  full = FALSE
)
```

Arguments

`n`	The number of individuals per generation. If scalar, the number of generations `G >= 2` must also be specified. Otherwise, the length of `n` is the number of generations.

G	The number of generations (optional). Note G == 1 is founders only, so it is invalid (there is no pedigree). Must specify a G >= 2 if n is a scalar. If both G is specified and length(n) > 1, both values must agree.
sex	The numeric sex values for the founders (1L for male, 2L for female). By default they are drawn randomly using draw_sex().
kinship_local	The local kinship matrix of the founder population. The default value is half the identity matrix, which corresponds to locally unrelated and locally outbred founders. This "local" kinship is the basis for all kinship calculations used to decide on close relative avoidance. The goal is to make a decision to not pair close relatives based on the pedigree only (and not based on population structure, which otherwise increases all kinship values), so the default value is appropriate.
cutoff	Local kinship values strictly less than cutoff are required for pairs. The default value of 1/4^3 corresponds to second cousins, so those and closer relatives are forbidden pairs (but a third cousin pair is allowed).
children_min	The minimum number of children per family. Must be 0 or larger, but not exceed the average number of children per family in each generation (varies depending on how many individuals were left unpaired, but this upper limit is approximately 2 * n[i] / n[i-1] for generation i). The number of children for each given family is first chosen as children_min plus a Poisson random variable with parameter equal to the mean number of children per family needed to achieve the desired population size (n) minus children_min. As these numbers may not exactly equal the target population size, random families are incremented or decremented (respecting the minimum family size) by single counts until the target population size is met.
full	If TRUE, part of the return object is a list of local kinship matrices for every generation. If FALSE (default), only the local kinship matrix of the last generation is returned.

Value

A list with these named elements:

- fam: the pedigree, a tibble in plink FAM format. Following the column naming convention of the related genio package, it contains columns:
  - fam: Family ID, trivial "fam1" for all individuals
  - id: Individual ID, in this case a code of format (in regular expression) "(\d+)-(\d+)" where the first integer is the generation number and the second integer is the index number (1 to n[g] for generation g).
  - pat: Paternal ID. Matches an id except for founders, which have fathers set to NA.
  - mat: Maternal ID. Matches an id except for founders, which have mothers set to NA.
  - sex: integers 1L (male) or 2L (female) which were drawn randomly; no other values occur in these outputs.
  - pheno: Phenotype, here all 0 (missing value).
- ids: a list of IDs for each generation (indexed in the list by generation).
- kinship_local: if full = FALSE, the local kinship matrix of the last generation, otherwise a list of local kinship matrices for every generation.

See Also

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

Examples

```
# number of individuals for each generation
n <- c(15, 20, 25)

# create random pedigree with 3 generations, etc
data <- sim_pedigree( n )

# this is the FAM table defining the entire pedigree,
# which is the most important piece of information desired!
data$fam

# the IDs separated by generation
data$ids

# bonus: the local kinship matrix of the final generation
data$kinship_local
```

Index