

Package ‘malan’

July 22, 2025

Type Package

Title MAle Lineage ANalysis

Version 1.0.3

Author Mikkel Meyer Andersen [aut, cre]

Maintainer Mikkel Meyer Andersen <mikl@math.aau.dk>

Description MAle Lineage ANalysis by simulating
genealogies backwards and imposing short tandem repeats (STR) mutations forwards.
Intended for forensic Y chromosomal STR (Y-STR) haplotype analyses.
Numerous analyses are possible, e.g. number of matches and meiotic distance to matches.
Refer to papers mentioned in citation("`malan") (DOI's:
<[doi:10.1371/journal.pgen.1007028](https://doi.org/10.1371/journal.pgen.1007028)>,
<[doi:10.21105/joss.00684](https://doi.org/10.21105/joss.00684)> and
<[doi:10.1016/j.fsigen.2018.10.004](https://doi.org/10.1016/j.fsigen.2018.10.004)>).

License GPL-2 | file LICENSE

Imports Rcpp (>= 0.12.7), RcppProgress (>= 0.2.1), RcppArmadillo (>= 0.9.880.1.0), igraph (>= 1.0.1), tibble (>= 1.1), magrittr (>= 1.5), methods

Depends R (>= 3.6), dplyr (>= 0.7.3), tidygraph (>= 1.0.0.9999)

LinkingTo Rcpp, RcppArmadillo, RcppProgress

Suggests knitr, rmarkdown, testthat, ggraph, dirmult, Rmpfr

BugReports <https://github.com/mikldk/malan/issues>

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-11-28 17:50:05 UTC

Contents

malan-package	4
analyse_mixture_result	5
analyse_mixture_results	5
as_tbl_graph.malan_pedigreelist	6
brothers_matching	7
build_haplotype_hashmap	7
build_pedigrees	8
calc_autosomal_genotype_conditional_cumdist	8
calc_autosomal_genotype_probs	9
construct_M	9
count_brothers	10
count_haplotype_near_matches_individuals	10
count_haplotype_occurrences_individuals	11
count_haplotype_occurrences_pedigree	12
count_uncles	12
delete_haplotypeids_hashmap	13
estimate_autotheta_1subpop_genotypes	13
estimate_autotheta_1subpop_individuals	14
estimate_autotheta_subpops_genotypes	15
estimate_autotheta_subpops_individuals	16
estimate_autotheta_subpops_pids	16
estimate_autotheta_subpops_unweighted_genotypes	17
estimate_autotheta_subpops_unweighted_pids	18
father_matches	18
from_igraph	19
from_igraph_rcpp	19
generate_get_founder_haplotype_db	20
generate_get_founder_haplotype_ladder	20
get_allele_counts_genotypes	21
get_allele_counts_pids	21
get_brothers	22
get_children	22
get_cousins	23
get_family_info	23
get_father	24
get_generation	24
get_haplotype	25
get_haplotypes_individuals	25
get_haplotypes_in_pedigree	26
get_haplotypes_pids	27
get_individual	27
get_individuals	28
get_matching_pids_from_hashmap	28
get_nodes_edges	29
get_pedigrees_tidy	29
get_pedigree_as_graph	30

get_pedigree_from_individual	30
get_pedigree_id	31
get_pedigree_id_from_pid	31
get_pid	32
get_pids_in_pedigree	32
get_uncles	33
get_zero_haplotype_generator	33
grandfather_matches	34
haplotypes_to_hashes	34
haplotype_matches_individuals	35
haplotype_partially_matches_individuals	35
infer_generation	36
infer_generations	36
load_haplotypes	37
load_individuals	37
meioses_generation_distribution	38
meiotic_dist	38
meiotic_dist_threshold	39
meiotic_radius	39
mixture_info_by_individuals_2pers	40
mixture_info_by_individuals_3pers	40
mixture_info_by_individuals_4pers	41
mixture_info_by_individuals_5pers	42
pedigrees_all_populate_autosomal	43
pedigrees_all_populate_haplotypes	44
pedigrees_all_populate_haplotypes_custom_founders	45
pedigrees_all_populate_haplotypes_ladder_bounded	46
pedigrees_count	47
pedigrees_table	48
pedigree_as_igraph	48
pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists	49
pedigree_haplotype_near_matches_meiosis	50
pedigree_size	50
pedigree_size_generation	51
plot.malan_pedigree	51
plot.malan_pedigreelist	52
population_populate_autosomal_infinite_alleles	53
population_size_generation	53
print.malan_pedigree	54
print.malan_pedigreelist	54
print.malan_population	55
print.malan_population_abort	55
print_individual	56
relationship_allele_diff_dist	56
relationship_allele_diff_dist_sym	57
sample_autosomal_genotype	57
sample_geneology	58
sample_geneology_varying_size	60

set_generation	62
split_by_haplotypes	62
test_create_population	63
ystr_kits	63
ystr_markers	64
[[.malan_pedigreelist	64
[[.malan_population	65
Index	66

malan-package	<i>MAle Lineage ANalysis</i>
---------------	------------------------------

Description

Simulating genealogies backwards and imposing STR mutations forwards.

Details

See vignettes and manual for documentation.

Disclaimer: THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Author(s)

Maintainer: Mikkel Meyer Andersen <mikl@math.aau.dk>

References

Andersen MM, Balding DJ (2017) How convincing is a matching Y-chromosome profile? PLoS Genet 13(11): e1007028. doi:10.1371/journal.pgen.1007028.

See Also

Useful links:

- Report bugs at <https://github.com/mikldk/malan/issues>

`analyse_mixture_result`*Analyse mixture results*

Description

Calculate LR-like quantities by haplotype counts.

Usage

```
analyse_mixture_result(  
  mix_res,  
  unique_haps_in_mixture,  
  unique_haps_in_mixture_counts  
)
```

Arguments

<code>mix_res</code>	Mixture result from mixture_info_by_individuals_2pers() , mixture_info_by_individuals_3pers() , mixture_info_by_individuals_4pers() , mixture_info_by_individuals_5pers()
<code>unique_haps_in_mixture</code>	Included unique haplotypes to use as elements in contributor sets.
<code>unique_haps_in_mixture_counts</code>	Population counts of the included haplotypes

Details

NOTE: Only takes up to 9 contributors!

Value

A list with numeric quantities

`analyse_mixture_results`*Analyse mixture results in a vectorised fashion*

Description

Refer to [analyse_mixture_result\(\)](#) for details. Essentially, [analyse_mixture_result\(\)](#) is run on each element of `mixture_results`.

Usage

```
analyse_mixture_results(
  mixture_results,
  unique_haps_in_mixture_list,
  unique_haps_in_mixture_counts_list
)
```

Arguments

`mixture_results`
List of n mixture results from `mixture_info_by_individuals_2pers()`, `mixture_info_by_individuals_3pers()`, `mixture_info_by_individuals_4pers()`, `mixture_info_by_individuals_5pers()`

`unique_haps_in_mixture_list`
List of n included unique haplotypes, one for each element in `mix_res`

`unique_haps_in_mixture_counts_list`
List of n population counts of the included unique haplotypes

Details

NOTE: Only takes up to 9 contributors!

Value

A list with lists of numeric quantities

```
as_tbl_graph.malan_pedigreelist
  Get tidy graph object
```

Description

Get tidy graph object `tbl_graph()`, e.g. to plot with `ggraph()`.

Usage

```
## S3 method for class 'malan_pedigreelist'
as_tbl_graph(x, ...)
```

Arguments

`x` malan_pedigreelist

`...` Ignored

Value

`tbl_graph()` object

brothers_matching	<i>Number of brothers with matching haplotype</i>
-------------------	---

Description

Get individual's number of brothers that matches individual's haplotype

Usage

```
brothers_matching(individual)
```

Arguments

individual	individual
------------	------------

Value

Number of brothers that matches individual's haplotype

build_haplotype_hashmap	<i>Build hashmap of haplotype to individuals</i>
-------------------------	--

Description

Makes it possible to find all individuals' pid with a certain haplotype. Must be used with e.g. [get_matching_pids_from_hashmap\(\)](#).

Usage

```
build_haplotype_hashmap(individuals, progress = TRUE)
```

Arguments

individuals	List of individuals to build hashmap of
progress	Show progress?

Value

External pointer to hashmap with haplotype as keys and vector of individuals' pid as value

See Also

[get_matching_pids_from_hashmap\(\)](#).

build_pedigrees	<i>Build pedigrees from (individuals in) a population.</i>
-----------------	--

Description

In a newly simulated population, each individual only knows its father and children. Using this information, this function builds pedigrees. This makes it easier to e.g. population haplotypes, find path between two individuals (if they are not in the same pedigree, they are not connected).

Usage

```
build_pedigrees(population, progress = TRUE)
```

Arguments

population	Population generated by sample_geneology() or sample_geneology_varying_size() .
progress	Show progress.

Value

An object with class malan_pedigreelist (an internal list of external pointers to pedigrees).

See Also

[sample_geneology\(\)](#) and [sample_geneology_varying_size\(\)](#) for simulating populations.

Examples

```
sim <- sample_geneology(100, 10)
str(sim, 1)
sim$population
peds <- build_pedigrees(sim$population)
peds
```

calc_autosomal_genotype_conditional_cumdist	<i>Calculate conditional genotype cumulative probabilities with theta</i>
---	---

Description

Calculate conditional genotype cumulative probabilities with theta

Usage

```
calc_autosomal_genotype_conditional_cumdist(allele_dist, theta)
```

Arguments

allele_dist	Allele distribution (probabilities) – gets normalised
theta	Theta correction between 0 and 1 (both included)

Value

Matrix: row i: conditional cumulative distribution of alleles given allele i

`calc_autosomal_genotype_probs`*Calculate genotype probabilities with theta*

Description

Calculate genotype probabilities with theta

Usage

```
calc_autosomal_genotype_probs(allele_dist, theta)
```

Arguments

allele_dist	Allele distribution (probabilities) – gets normalised
theta	Theta correction between 0 and 1 (both included)

`construct_M`*Construct M matrix*

Description

Construct M matrix

Usage

```
construct_M(meioses, mu_dw, mu_up)
```

Arguments

meioses	number of meioses separating the two individuals
mu_dw	mutation rate for 1-step down-mutation
mu_up	mutation rate for 1-step up-mutation

count_brothers	<i>Number of brothers</i>
----------------	---------------------------

Description

Get individual's number of brothers

Usage

```
count_brothers(individual)
```

Arguments

individual	individual
------------	------------

Value

Number of brothers

See Also

[get_brothers\(\)](#)

count_haplotype_near_matches_individuals
<i>Count near haplotype matches in list of individuals</i>

Description

Counts the number of types close to haplotype in individuals.

Usage

```
count_haplotype_near_matches_individuals(individuals, haplotype, max_dist)
```

Arguments

individuals	List of individuals to count occurrences in.
haplotype	Haplotype to count near-matches occurrences of.
max_dist	Maximum distance (0 = match, 1 = 1 STR allele difference, ...)

Value

Number of times that a haplotype within a radius of max_dist of haplotype occurred amongst individuals.

See Also

[count_haplotype_occurrences_individuals\(\)](#), [pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists\(\)](#)

count_haplotype_occurrences_individuals

Count haplotypes occurrences in list of individuals

Description

Counts the number of types haplotype appears in individuals.

Usage

```
count_haplotype_occurrences_individuals(individuals, haplotype)
```

Arguments

individuals	List of individuals to count occurrences in.
haplotype	Haplotype to count occurrences of.

Value

Number of times that haplotype occurred amongst individuals.

See Also

[pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists\(\)](#), [count_haplotype_near_matches_individuals\(\)](#)

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_all_populate_haplotypes(peds, 2, c(0, 0))
count_haplotype_occurrences_individuals(sim$end_generation_individuals, c(0, 0))
```

count_haplotype_occurrences_pedigree
Count haplotypes occurrences in pedigree

Description

Counts the number of types haplotype appears in pedigree.

Usage

```
count_haplotype_occurrences_pedigree(
  pedigree,
  haplotype,
  generation_upper_bound_in_result = -1L
)
```

Arguments

pedigree Pedigree to count occurrences in.
 haplotype Haplotype to count occurrences of.
 generation_upper_bound_in_result
 Only consider matches in generation 0, 1, ... generation_upper_bound_in_result.
 -1 means disabled, consider all generations. End generation is generation 0.
 Second last generation is 1. And so on.

Value

Number of times that haplotype occurred in pedigree.

See Also

[pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists\(\)](#).

count_uncles *Number of uncles*

Description

Get individual's number of uncles

Usage

```
count_uncles(individual)
```

Arguments

individual individual

Value

Number of uncles

See Also

[get_uncles\(\)](#)

delete_haplotypeids_hashmap
Delete haplotype hashmap

Description

Delete hashmap made by [build_haplotype_hashmap\(\)](#).

Usage

```
delete_haplotypeids_hashmap(hashmap)
```

Arguments

hashmap Hashmap made by [build_haplotype_hashmap\(\)](#)

See Also

[get_matching_pids_from_hashmap\(\)](#) and [build_haplotype_hashmap\(\)](#).

estimate_autotheta_1subpop_genotypes
Estimate autosomal theta from genotypes

Description

Estimate autosomal theta for one subpopulation given a sample of genotypes.

Usage

```
estimate_autotheta_1subpop_genotypes(genotypes, return_estimation_info = FALSE)
```

Arguments

genotypes Matrix of genotypes: two columns (allele1 and allele2) and a row per individual
 return_estimation_info Whether to return the quantities used to estimate theta

Details

Assumes that `pedigrees_all_populate_autosomal()` was used first to populate autosomal genotypes.

Value

List:

- theta
 - estimate: Vector of length 1 containing estimate of theta or NA if it could not be estimated
 - error: true if an error happened, false otherwise
 - details: contains description if an error happened
 - estimation_info: If return_estimation_info = true: a list with information used to estimate theta. Else NULL.

estimate_autotheta_1subpop_individuals

Estimate autosomal theta from individuals

Description

Estimate autosomal theta for one subpopulation given a list of individuals.

Usage

```
estimate_autotheta_1subpop_individuals(
  individuals,
  return_estimation_info = FALSE
)
```

Arguments

individuals Individuals to get haplotypes for.
 return_estimation_info Whether to return the quantities used to estimate theta

Details

Assumes that `pedigrees_all_populate_autosomal()` was used first to populate autosomal genotypes.

Value

List:

- theta
 - estimate: Vector of length 1 containing estimate of theta or NA if it could not be estimated
 - error: true if an error happened, false otherwise
 - details: contains description if an error happened
 - estimation_info: If return_estimation_info = true: a list with information used to estimate theta. Else NULL.

 estimate_autotheta_subpops_genotypes

Estimate autosomal F, theta, and f from subpopulations of genotypes

Description

Estimates autosomal F, theta, and f for a number of subpopulations given a list of genotypes.

Usage

```
estimate_autotheta_subpops_genotypes(subpops, subpops_sizes)
```

Arguments

subpops	List of subpopulations, each a list of individuals
subpops_sizes	Size of each subpopulation

Details

Assumes that [pedigrees_all_populate_autosomal\(\)](#) was used first to populate autosomal genotypes.

Based on Bruce S Weir, Genetic Data Analysis 2, 1996. (GDA2).

Value

Estimates of autosomal F, theta, and f as well as additional information

```
estimate_autotheta_subpops_individuals
```

Estimate autosomal F, theta, and f from subpopulations of individuals

Description

Estimates autosomal F, theta, and f for a number of subpopulations given a list of individuals.

Usage

```
estimate_autotheta_subpops_individuals(subpops, subpops_sizes)
```

Arguments

subpops	List of subpopulations, each a list of individuals
subpops_sizes	Size of each subpopulation

Details

Assumes that [pedigrees_all_populate_autosomal\(\)](#) was used first to populate autosomal genotypes.

Based on Bruce S Weir, Genetic Data Analysis 2, 1996. (GDA2).

Value

Estimates of autosomal F, theta, and f as well as additional information

```
estimate_autotheta_subpops_pids
```

Estimate autosomal F, theta, and f from subpopulations of individual ids

Description

Estimates autosomal F, theta, and f for a number of subpopulations given a list of pids (individual ids).

Usage

```
estimate_autotheta_subpops_pids(population, subpops, subpops_sizes)
```

Arguments

population	Population obtain from simulation
subpops	List of individual pids
subpops_sizes	Size of each subpopulation

Details

Assumes that `pedigrees_all_populate_autosomal()` was used first to populate autosomal genotypes.

Based on Bruce S Weir, Genetic Data Analysis 2, 1996. (GDA2).

Value

Estimates of autosomal F, theta, and f as well as additional information

`estimate_autotheta_subpops_unweighted_genotypes`

Unweighted estimate of autosomal theta from subpopulations of genotypes

Description

Estimates unweighted autosomal theta for a number of subpopulations given a list of subpopulations of genotypes.

Usage

```
estimate_autotheta_subpops_unweighted_genotypes(subpops, assume_HWE)
```

Arguments

<code>subpops</code>	List of individual genotypes
<code>assume_HWE</code>	if the alleles themselves are used instead of genotypes

Details

Assumes that `pedigrees_all_populate_autosomal()` was used first to populate autosomal genotypes.

Based on Weir and Goudet, Genetics 2017: <http://www.genetics.org/content/early/2017/05/26/genetics.116.198424>

Value

Estimate of autosomal theta

<code>estimate_autotheta_subpops_unweighted_pids</code>
<i>Unweighted estimate of autosomal theta from subpopulations of individual ids</i>

Description

Estimates unweighted autosomal theta for a number of subpopulations given a list of pids (individual ids).

Usage

`estimate_autotheta_subpops_unweighted_pids(population, subpops, assume_HWE)`

Arguments

- | | |
|-------------------------|---|
| <code>population</code> | Population obtain from simulation |
| <code>subpops</code> | List of individual pids |
| <code>assume_HWE</code> | if the alleles themselves are used instead of genotypes |

Details

Assumes that `pedigrees_all_populate_autosomal()` was used first to populate autosomal genotypes.
Based on Weir and Goudet, Genetics 2017: <http://www.genetics.org/content/early/2017/05/26/genetics.116.198424>

Value

Estimate of autosomal theta

<code>father_matches</code>	<i>Father matches</i>
-----------------------------	-----------------------

Description

Does the father have the same profile as individual?

Usage

`father_matches(individual)`

Arguments

- | | |
|-------------------------|------------|
| <code>individual</code> | individual |
|-------------------------|------------|

Value

Whether father has the same profile as individual or not

from_igraph	<i>Convert igraph to population</i>
-------------	-------------------------------------

Description

Convert igraph to population

Usage

```
from_igraph(x, ...)
```

Arguments

x	igraph, must be a forest of directed trees with unique positive integer names (as they will be pid's)
...	Ignored

Value

A population

Examples

```
g <- igraph::graph_from_literal( 2 +- 1 -+ 3, 4 -+ 5 )
plot(g)
pop <- from_igraph(g)
peds <- build_pedigrees(pop, progress = FALSE)
plot(peds)
infer_generations(peds)
get_generation(get_individual(pop, 1))
get_generation(get_individual(pop, 2))
get_generation(get_individual(pop, 3))
get_generation(get_individual(pop, 4))
get_generation(get_individual(pop, 5))
```

from_igraph_rcpp	<i>Generate paternal brothers population</i>
------------------	--

Description

Generate paternal brothers population

Usage

```
from_igraph_rcpp(vertices, edges)
```

Arguments

vertices vector of vertices
edges matrix with edges

Value

An external pointer to the population.

generate_get_founder_haplotype_db
<i>Generate a function to simulate pedigree founder haplotype based on a haplotype database</i>

Description

Generate a function to simulate pedigree founder haplotype based on a haplotype database

Usage

generate_get_founder_haplotype_db(db)

Arguments

db data frame or matrix with haplotypes from which the founder is randomly simulated

generate_get_founder_haplotype_ladder
<i>Generate a function to simulate pedigree founder haplotype based on ladder information</i>

Description

Generate a function to simulate pedigree founder haplotype based on ladder information

Usage

generate_get_founder_haplotype_ladder(ladder_min, ladder_max)

Arguments

ladder_min vector of minimum alleles; ladder_min[i] is the minimum allele at locus i
ladder_max vector of minimum alleles; ladder_max[i] is the maximum allele at locus i

`get_allele_counts_genotypes`*Get autosomal allele counts from subpopulations of genotypes*

Description

Assumes that `pedigrees_all_populate_autosomal()` was used first to populate autosomal genotypes.

Usage

```
get_allele_counts_genotypes(subpops)
```

Arguments

subpops	List of individual genotypes
---------	------------------------------

Value

Matrix with allele counts

`get_allele_counts_pids`*Get autosomal allele counts from subpopulations given by pids*

Description

Assumes that `pedigrees_all_populate_autosomal()` was used first to populate autosomal genotypes.

Usage

```
get_allele_counts_pids(population, subpops)
```

Arguments

population	Population obtain from simulation
subpops	List of individual pids

Value

Matrix with allele counts

<code>get_brothers</code>	<i>Get brothers</i>
---------------------------	---------------------

Description

Get individual’s brothers

Usage

`get_brothers(individual)`

Arguments

`individual` `individual`

Value

List with brothers

See Also

[get_father\(\)](#), [get_uncles\(\)](#), [get_children\(\)](#), [get_cousins\(\)](#)

<code>get_children</code>	<i>Get children</i>
---------------------------	---------------------

Description

Get individual’s children

Usage

`get_children(individual)`

Arguments

`individual` `individual`

Value

List with children

See Also

[get_father\(\)](#), [get_brothers\(\)](#), [get_uncles\(\)](#), [get_cousins\(\)](#)

get_cousins	<i>Get cousins</i>
-------------	--------------------

Description

Get individual's cousins

Usage

```
get_cousins(individual)
```

Arguments

individual	individual
------------	------------

Value

List with cousins

See Also

[get_brothers\(\)](#), [get_uncles\(\)](#), [get_children\(\)](#)

get_family_info	<i>Get individual's family information</i>
-----------------	--

Description

Get individual's family information

Usage

```
get_family_info(individual)
```

Arguments

individual	individual
------------	------------

Value

List with family information

get_father	<i>Get father</i>
------------	-------------------

Description

Get individual's father

Usage

```
get_father(individual)
```

Arguments

individual individual

Value

Father

See Also

[get_brothers\(\)](#), [get_uncles\(\)](#), [get_children\(\)](#), [get_cousins\(\)](#)

get_generation	<i>Get individual's generation number</i>
----------------	---

Description

Note that generation 0 is final, end generation. 1 is second last generation etc.

Usage

```
get_generation(individual)
```

Arguments

individual Individual

Value

generation

Examples

```
sim <- sample_geneology(100, 10)
indv <- get_individual(sim$population, 1)
get_generation(indv)
```

get_haplotype	<i>Get haplotype from an individual</i>
---------------	---

Description

Requires that haplotypes are first populated, e.g. with [pedigrees_all_populate_haplotypes\(\)](#), [pedigrees_all_populate_haplotypes_custom_founders\(\)](#), or [pedigrees_all_populate_haplotypes_ladder_boundaries\(\)](#).

Usage

```
get_haplotype(individual)
```

Arguments

`individual` Individual to get haplotypes for.

Value

Haplotype for individual.

See Also

[get_haplotypes_individuals\(\)](#) and [get_haplotypes_pids\(\)](#).

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_all_populate_haplotypes(peds, 2, c(1, 1))
get_haplotype(sim$end_generation_individuals[[1]])
```

get_haplotypes_individuals	<i>Get haplotype matrix from list of individuals</i>
----------------------------	--

Description

Requires that haplotypes are first populated, e.g. with [pedigrees_all_populate_haplotypes\(\)](#), [pedigrees_all_populate_haplotypes_custom_founders\(\)](#), or [pedigrees_all_populate_haplotypes_ladder_boundaries\(\)](#).

Usage

```
get_haplotypes_individuals(individuals)
```

Arguments

individuals Individuals to get haplotypes for.

Value

Matrix of haplotypes where row *i* is the haplotype of individuals[[*i*]].

See Also

[get_haplotypes_pids\(\)](#).

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_all_populate_haplotypes(peds, 2, c(1, 1))
get_haplotypes_individuals(sim$end_generation_individuals)
```

get_haplotypes_in_pedigree

Get haplotypes in pedigree

Description

Get haplotypes in pedigree

Usage

```
get_haplotypes_in_pedigree(ped)
```

Arguments

ped Pedigree

Value

List with haplotypes

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_all_populate_haplotypes(peds, 2, c(1, 1))
get_haplotypes_in_pedigree(peds[[1]])
```

get_haplotypes_pids	<i>Get haplotypes from a vector of pids.</i>
---------------------	--

Description

Requires that haplotypes are first populated, e.g. with [pedigrees_all_populate_haplotypes\(\)](#), [pedigrees_all_populate_haplotypes_custom_founders\(\)](#), or [pedigrees_all_populate_haplotypes_ladder_boundaries\(\)](#).

Usage

```
get_haplotypes_pids(population, pids)
```

Arguments

population	Population
pids	Vector of pids to get haplotypes for.

Value

Matrix of haplotypes where row *i* is the haplotype of individuals[[*i*]].

See Also

[get_haplotypes_individuals\(\)](#).

get_individual	<i>Get individual by pid</i>
----------------	------------------------------

Description

Get individual by pid

Usage

```
get_individual(population, pid)
```

Arguments

population	Population
pid	pid

Value

Individual

Examples

```
sim <- sample_geneology(100, 10)
indv <- get_individual(sim$population, 1)
get_pid(indv)
```

get_individuals	<i>Get all individuals in population</i>
-----------------	--

Description

Get all individuals in population

Usage

```
get_individuals(population)
```

Arguments

population	Population
------------	------------

get_matching_pids_from_hashmap	<i>Get individuals with a certain haplotype id by hashmap lookup</i>
--------------------------------	--

Description

By using hashmap made by [build_haplotype_hashmap\(\)](#), it is easy to get all individuals with a certain haplotype id.

Usage

```
get_matching_pids_from_hashmap(hashmap, haplotype)
```

Arguments

hashmap	Hashmap to make lookup in, made by build_haplotype_hashmap()
haplotype	to get individuals that has this haplotype id

Value

List of individuals with a given haplotype id

See Also

[build_haplotype_hashmap\(\)](#).

get_nodes_edges	<i>Get nodes and edges</i>
-----------------	----------------------------

Description

Get nodes and edges in malan_pedigreelist. For example to plot via [as_tbl_graph\(\)](#).

Usage

```
get_nodes_edges(x, ...)
```

Arguments

x	malan_pedigreelist
...	Ignored

Value

List with entries nodes and edges

get_pedigrees_tidy	<i>Get pedigrees information in tidy format</i>
--------------------	---

Description

Get pedigrees information in tidy format

Usage

```
get_pedigrees_tidy(pedigrees)
```

Arguments

pedigrees	Pedigrees
-----------	-----------

get_pedigree_as_graph *Get pedigree information as graph (mainly intended for plotting)*

Description

Get pedigree information as graph (mainly intended for plotting)

Usage

```
get_pedigree_as_graph(ped)
```

Arguments

ped	Pedigree
-----	----------

get_pedigree_from_individual
Get pedigree from individual

Description

Get pedigree from individual

Usage

```
get_pedigree_from_individual(individual)
```

Arguments

individual	Individual
------------	------------

Value

pedigree

get_pedigree_id	<i>Get pedigree id</i>
-----------------	------------------------

Description

Get pedigree id

Usage

```
get_pedigree_id(ped)
```

Arguments

ped	Pedigree
-----	----------

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
get_pedigree_id(peds[[1]])
```

get_pedigree_id_from_pid	<i>Get pedigree ids from pids</i>
--------------------------	-----------------------------------

Description

Get pedigree ids from pids

Usage

```
get_pedigree_id_from_pid(population, pids)
```

Arguments

population	Population
pids	Pids

Value

Vector with pedigree ids

get_pid	<i>Get pid from individual</i>
---------	--------------------------------

Description

Get pid from individual

Usage

```
get_pid(individual)
```

Arguments

individual	Individual to get pid of
------------	--------------------------

Value

pid

Examples

```
sim <- sample_geneology(100, 10)
indv <- get_individual(sim$population, 1)
get_pid(indv)
```

get_pids_in_pedigree	<i>Get pids in pedigree</i>
----------------------	-----------------------------

Description

Get pids in pedigree

Usage

```
get_pids_in_pedigree(ped)
```

Arguments

ped	Pedigree
-----	----------

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
get_pids_in_pedigree(peds[[1]])
```

`get_uncles`*Get uncles*

Description

Get individual's uncles

Usage

```
get_uncles(individual)
```

Arguments

`individual` `individual`

Value

List with uncles

See Also

[get_brothers\(\)](#), [get_children\(\)](#), [get_cousins\(\)](#)

`get_zero_haplotype_generator`*Generate a function to generate the zero haplotype*

Description

Generate a function to generate the zero haplotype

Usage

```
get_zero_haplotype_generator(loci)
```

Arguments

`loci` Number of loci

grandfather_matches	<i>Grandfather matches</i>
---------------------	----------------------------

Description

Does the grandfather have the same profile as individual?

Usage

```
grandfather_matches(individual)
```

Arguments

individual	individual
------------	------------

Value

Whether grandfather has the same profile as individual or not

haplotypes_to_hashes	<i>Convert haplotypes to hashes (integers)</i>
----------------------	--

Description

Individuals with the same haplotype will have the same hash (integer) and individuals with different haplotypes will have different hashes (integers).

Usage

```
haplotypes_to_hashes(population, pids)
```

Arguments

population	Population obtained from simulation
pids	Vector of individual pids

Details

This can be useful if for example using haplotypes to define groups and the haplotype itself is not of interest.

Value

Integer vector with haplotype hashes

`haplotype_matches_individuals`*Get individuals matching from list of individuals*

Description

Get the individuals that matches haplotype in individuals.

Usage

```
haplotype_matches_individuals(individuals, haplotype)
```

Arguments

<code>individuals</code>	List of individuals to count occurrences in.
<code>haplotype</code>	Haplotype to count occurrences of.

Value

List of individuals that matches haplotype amongst individuals.

See Also

[pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists\(\)](#).

`haplotype_partially_matches_individuals`*Get individuals partially matching from list of individuals*

Description

Get the individuals that partially matches haplotype in individuals.

Usage

```
haplotype_partially_matches_individuals(  
  individuals,  
  haplotype,  
  ignore_loci = as.integer(c())  
)
```

Arguments

<code>individuals</code>	List of individuals to count occurrences in.
<code>haplotype</code>	Haplotype to count occurrences of.
<code>ignore_loci</code>	Vector of loci to ignore (1 = ignore first locus etc.)

Value

List of individuals that partially matches haplotype amongst individuals.

infer_generation	<i>Infer individual's generation number</i>
------------------	---

Description

Takes as input final generation, then moves up in pedigree and increments generation number.

Usage

```
infer_generation(final_generation)
```

Arguments

final_generation	Individuals in final generation
------------------	---------------------------------

Details

Note: Only works when all final generation individuals are provided.

infer_generations	<i>Infer generation numbers from pedigrees</i>
-------------------	--

Description

Infer generation numbers from pedigrees

Usage

```
infer_generations(peds)
```

Arguments

peds	Pedigrees inferred by build_pedigrees()
------	---

Value

Nothing

load_haplotypes	<i>Load haplotypes to individuals</i>
-----------------	---------------------------------------

Description

Note that individuals loaded this way does not have information about generation.

Usage

```
load_haplotypes(population, pid, haplotypes, progress = TRUE)
```

Arguments

population	of individuals
pid	ID of male
haplotypes	• row i has pid[i] ID
progress	Show progress.

load_individuals	<i>Construct a population from data</i>
------------------	---

Description

Note that individuals loaded this way does not have information about generation.

Usage

```
load_individuals(pid, pid_dad, progress = TRUE, error_on_pid_not_found = TRUE)
```

Arguments

pid	ID of male
pid_dad	ID of male's father, 0 if not known
progress	Show progress.
error_on_pid_not_found	Error if pid not found

meioses_generation_distribution	<i>Meiotic distribution</i>
---------------------------------	-----------------------------

Description

Get the distribution of number of meioses from individual to all individuals in individual's pedigree. Note the generation_upper_bound_in_result parameter.

Usage

```
meioses_generation_distribution(  
  individual,  
  generation_upper_bound_in_result = -1L  
)
```

Arguments

individual	Individual to calculate all meiotic distances from
generation_upper_bound_in_result	Limit on distribution; -1 means no limit. 0 is the final generation. 1 second last generation etc.

meiotic_dist	<i>Meiotic distance between two individuals</i>
--------------	---

Description

Get the number of meioses between two individuals. Note, that pedigrees must first have been inferred by [build_pedigrees\(\)](#).

Usage

```
meiotic_dist(ind1, ind2)
```

Arguments

ind1	Individual 1
ind2	Individual 2

Value

Number of meioses between ind1 and ind2 if they are in the same pedigree, else -1.

`meiotic_dist_threshold`*Meiotic distance between two individuals (with threshold)*

Description

Get the number of meioses between two individuals. Note, that pedigrees must first have been inferred by `build_pedigrees()`.

Usage

```
meiotic_dist_threshold(ind1, ind2, threshold)
```

Arguments

<code>ind1</code>	Individual 1
<code>ind2</code>	Individual 2
<code>threshold</code>	Max search radius, if exceeding, return -1

Value

Number of meioses between `ind1` and `ind2` if they are in the same pedigree, else -1.

`meiotic_radius`*Meiotic radius*

Description

Get all individual IDs within a meiotic radius Note, that pedigrees must first have been inferred by `build_pedigrees()`.

Usage

```
meiotic_radius(ind, radius)
```

Arguments

<code>ind</code>	Individual
<code>radius</code>	Max radius

Value

Matrix with ID and meiotic radius

mixture_info_by_individuals_2pers

Mixture information about 2 persons' mixture of donor1 and donor2.

Description

Mixture information about 2 persons' mixture of donor1 and donor2.

Usage

```
mixture_info_by_individuals_2pers(
  individuals,
  donor1,
  donor2,
  include_genealogy_info = TRUE
)
```

Arguments

individuals	Individuals to consider as possible contributors and thereby get information from.
donor1	Contributor1/donor 1
donor2	Contributor2/donor 2
include_genealogy_info	Include information about meiotic distances and family info

Value

A list with mixture information about the mixture donor1+donor2+donor3 from individuals

See Also

[mixture_info_by_individuals_3pers](#), [mixture_info_by_individuals_4pers](#), [mixture_info_by_individuals_5pers](#)

mixture_info_by_individuals_3pers

Mixture information about 3 persons' mixture of donor1, donor2 and donor3.

Description

Mixture information about 3 persons' mixture of donor1, donor2 and donor3.

Usage

```
mixture_info_by_individuals_3pers(individuals, donor1, donor2, donor3)
```

Arguments

individuals	Individuals to consider as possible contributors and thereby get information from.
donor1	Contributor1/donor 1
donor2	Contributor2/donor 2
donor3	Contributor3/donor 3

Value

A list with mixture information about the mixture donor1+donor2+donor3 from individuals

See Also

[mixture_info_by_individuals_2pers](#), [mixture_info_by_individuals_4pers](#), [mixture_info_by_individuals_5pers](#)

mixture_info_by_individuals_4pers	<i>Mixture information about 4 persons' mixture of donor1, donor2, donor3 and donor4.</i>
-----------------------------------	---

Description

Mixture information about 4 persons' mixture of donor1, donor2, donor3 and donor4.

Usage

```
mixture_info_by_individuals_4pers(individuals, donor1, donor2, donor3, donor4)
```

Arguments

individuals	Individuals to consider as possible contributors and thereby get information from.
donor1	Contributor1/donor 1
donor2	Contributor2/donor 2
donor3	Contributor3/donor 3
donor4	Contributor4/donor 4

Value

A list with mixture information about the mixture donor1+donor2+donor3 from individuals

See Also

[mixture_info_by_individuals_2pers](#), [mixture_info_by_individuals_3pers](#), [mixture_info_by_individuals_5pers](#)

mixture_info_by_individuals_5pers
<i>Mixture information about 5 persons' mixture of donor1, donor2, donor3, donor4 and donor5.</i>

Description

Mixture information about 5 persons' mixture of donor1, donor2, donor3, donor4 and donor5.

Usage

```
mixture_info_by_individuals_5pers(  
  individuals,  
  donor1,  
  donor2,  
  donor3,  
  donor4,  
  donor5  
)
```

Arguments

individuals	Individuals to consider as possible contributors and thereby get information from.
donor1	Contributor1/donor 1
donor2	Contributor2/donor 2
donor3	Contributor3/donor 3
donor4	Contributor4/donor 4
donor5	Contributor5/donor 5

Value

A list with mixture information about the mixture donor1+donor2+donor3 from individuals

See Also

[mixture_info_by_individuals_2pers](#), [mixture_info_by_individuals_3pers](#), [mixture_info_by_individuals_4pers](#)

`pedigrees_all_populate_autosomal`*Populate 1-locus autosomal DNA profile in pedigrees with single-step mutation model.*

Description

Populate 1-locus autosomal DNA profile from founder and down in all pedigrees. Note, that only alleles from ladder is assigned and that all founders draw type randomly.

Usage

```
pedigrees_all_populate_autosomal(  
  pedigrees,  
  allele_dist,  
  theta,  
  mutation_rate,  
  progress = TRUE  
)
```

Arguments

<code>pedigrees</code>	Pedigree list in which to populate genotypes
<code>allele_dist</code>	Allele distribution (probabilities) – gets normalised
<code>theta</code>	Theta correction between 0 and 1 (both included)
<code>mutation_rate</code>	Mutation rate between 0 and 1 (both included)
<code>progress</code>	Show progress

Details

Note, that pedigrees must first have been inferred by [build_pedigrees\(\)](#).

See Also

[pedigrees_all_populate_haplotypes_custom_founders\(\)](#) and [pedigrees_all_populate_haplotypes_ladder_bour](#)

pedigrees_all_populate_haplotypes

Populate haplotypes in pedigrees (0-founder/unbounded).

Description

Populate haplotypes from founder and down in all pedigrees. Note, that haplotypes are unbounded and that all founders get haplotype `rep(0L, loci)`.

Usage

```
pedigrees_all_populate_haplotypes(
  pedigrees,
  loci,
  mutation_rates,
  prob_two_step = 0,
  prob_genealogical_error = 0,
  progress = TRUE
)
```

Arguments

<code>pedigrees</code>	Pedigree list in which to populate haplotypes
<code>loci</code>	Number of loci
<code>mutation_rates</code>	Vector with mutation rates, length <code>loci</code>
<code>prob_two_step</code>	Given a mutation happens, this is the probability that the mutation is a two-step mutation
<code>prob_genealogical_error</code>	Probability that a genealogical error happens: if so, give individual haplotype <code>rep(0L, loci)</code> instead of father's
<code>progress</code>	Show progress

Details

Note, that pedigrees must first have been inferred by [build_pedigrees\(\)](#).

See Also

[pedigrees_all_populate_haplotypes_custom_founders\(\)](#) and [pedigrees_all_populate_haplotypes_ladder_boundaries\(\)](#)

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_all_populate_haplotypes(peds, 2, c(1, 1))
get_haplotype(sim$end_generation_individuals[[1]])
```

pedigrees_all_populate_haplotypes_custom_founders

Populate haplotypes in pedigrees (custom founder/unbounded).

Description

Populate haplotypes from founder and down in all pedigrees. Note, that haplotypes are unbounded. All founders get a haplotype from calling the user provided function `get_founder_haplotype()`.

Usage

```
pedigrees_all_populate_haplotypes_custom_founders(
  pedigrees,
  mutation_rates,
  get_founder_haplotype = NULL,
  prob_two_step = 0,
  prob_genealogical_error = 0,
  progress = TRUE
)
```

Arguments

<code>pedigrees</code>	Pedigree list in which to populate haplotypes
<code>mutation_rates</code>	Vector with mutation rates
<code>get_founder_haplotype</code>	Function taking no arguments returning a haplotype of length(<code>mutation_rates</code>)
<code>prob_two_step</code>	Given a mutation happens, this is the probability that the mutation is a two-step mutation
<code>prob_genealogical_error</code>	Probability that a genealogical error happens: if so, give individual haplotype <code>get_founder_haplotype()</code> instead of father's
<code>progress</code>	Show progress

Details

Note, that pedigrees must first have been inferred by [build_pedigrees\(\)](#).

See Also

[pedigrees_all_populate_haplotypes\(\)](#) and [pedigrees_all_populate_haplotypes_ladder_bounded\(\)](#).

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_all_populate_haplotypes_custom_founders(
  peds, c(1, 1), function(x) c(10, 10))
get_haplotype(sim$end_generation_individuals[[1]])
```

pedigrees_all_populate_haplotypes_ladder_bounded

Populate haplotypes in pedigrees (custom founder/bounded).

Description

Populate haplotypes from founder and down in all pedigrees. Note, that haplotypes are bounded by ladder_min and ladder_max. All founders get a haplotype from calling the user provided function get_founder_haplotype().

Usage

```
pedigrees_all_populate_haplotypes_ladder_bounded(
  pedigrees,
  mutation_rates,
  ladder_min,
  ladder_max,
  get_founder_haplotype = NULL,
  prob_two_step = 0,
  prob_genealogical_error = 0,
  progress = TRUE
)
```

Arguments

pedigrees	Pedigree list in which to populate haplotypes
mutation_rates	Vector with mutation rates
ladder_min	Lower bounds for haplotypes, same length as mutation_rates
ladder_max	Upper bounds for haplotypes, same length as mutation_rates; all entries must be strictly greater than ladder_min
get_founder_haplotype	Function taking no arguments returning a haplotype of length(mutation_rates)
prob_two_step	Given a mutation happens, this is the probability that the mutation is a two-step mutation; refer to details for information about behaviour around ladder boundaries
prob_genealogical_error	Probability that a genealogical error happens: if so, give individual haplotype get_founder_haplotype() instead of father's
progress	Show progress

Details

Given that a two step mutation should happen (probability specified by `prob_two_step`): With distances ≥ 2 to ladder bounds, mutations happen as usual. At distance = 0 or 1 to a ladder bound, the mutation is forced to move away from the boundary.

Note, that pedigrees must first have been inferred by `build_pedigrees()`.

See Also

`pedigrees_all_populate_haplotypes()` and `pedigrees_all_populate_haplotypes_custom_founders()`.

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_all_populate_haplotypes_ladder_bounded(
  peds, c(1, 1), c(0L, 0L), c(10L, 10L),
  function(x) c(10, 10))
get_haplotype(sim$end_generation_individuals[[1]])
```

pedigrees_count	<i>Get number of pedigrees</i>
-----------------	--------------------------------

Description

Get number of pedigrees

Usage

```
pedigrees_count(pedigrees)
```

Arguments

pedigrees	Pedigrees
-----------	-----------

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_count(peds)
```

pedigrees_table	<i>Get distribution of pedigree sizes</i>
-----------------	---

Description

Get distribution of pedigree sizes

Usage

```
pedigrees_table(pedigrees)
```

Arguments

pedigrees	Pedigrees
-----------	-----------

Examples

```
sim <- sample_geneology(100, 10)
peds <- build_pedigrees(sim$population)
pedigrees_table(peds)
```

pedigree_as_igraph	<i>Convert pedigree to igraph</i>
--------------------	-----------------------------------

Description

Convert pedigree to igraph

Usage

```
pedigree_as_igraph(x, ...)
```

Arguments

x	Pedigree
...	ignored

Value

igraph object

pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists
Information about matching individuals

Description

Gives information about all individuals in pedigree that matches an individual. Just as [count_haplotype_occurrences_individuals\(\)](#) counts the number of occurrences amongst a list of individuals, this gives detailed information about matching individuals in the pedigree, e.g. meiotic distances and maximum L1 distance on the path as some of these matches may have (back)mutations between in between them (but often this will be 0).

Usage

```
pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists(
    suspect,
    generation_upper_bound_in_result = -1L,
    error_on_no_haplotype = TRUE
)
```

Arguments

suspect	Individual that others must match the profile of.
generation_upper_bound_in_result	Only consider matches in generation 0, 1, ... generation_upper_bound_in_result. -1 means disabled, consider all generations. End generation is generation 0. Second last generation is 1. And so on.
error_on_no_haplotype	raise error or silently ignore individuals with no haplotype

Value

Matrix with information about matching individuals. Columns in order: meioses (meiotic distance to suspect), max_L1 (on the path between the matching individual and suspect, what is the maximum L1 distance between the suspect's profile and the profiles of the individuals on the path), pid (pid of matching individual)

See Also

[count_haplotype_occurrences_individuals\(\)](#).

pedigree_haplotype_near_matches_meiosis

Information about almost matching individuals

Description

Gives information about all individuals in pedigree that almost matches an individual. Just as `count_haplotype_near_matches_individuals()` counts the number of occurrences amongst a list of individuals, this gives detailed information about almost matching individuals in the pedigree: for now, the meiotic distances.

Usage

```
pedigree_haplotype_near_matches_meiosis(
  suspect,
  max_dist,
  generation_upper_bound_in_result = -1L
)
```

Arguments

<code>suspect</code>	Individual that others must match the profile of.
<code>max_dist</code>	Maximum distance (0 = match, 1 = 1 STR allele difference, ...)
<code>generation_upper_bound_in_result</code>	Only consider matches in generation 0, 1, ... <code>generation_upper_bound_in_result</code> . -1 means disabled, consider all generations. End generation is generation 0. Second last generation is 1. And so on.

Value

Matrix with information about matching individuals. Columns in order: 1) meioses (meiotic distance to suspect), 2) haplotype distance, 3) pid (pid of matching individual)

See Also

`count_haplotype_near_matches_individuals()`.

pedigree_size

Get pedigree size

Description

Get pedigree size

Usage

```
pedigree_size(ped)
```

Arguments

ped	Pedigree
-----	----------

Examples

```
sim <- sample_genealogy(100, 10)
peds <- build_pedigrees(sim$population)
pedigree_size(peds[[1]])
```

pedigree_size_generation	<i>Size of pedigree</i>
--------------------------	-------------------------

Description

Get the size of the pedigree. Note the generation_upper_bound_in_result parameter.

Usage

```
pedigree_size_generation(pedigree, generation_upper_bound_in_result = -1L)
```

Arguments

pedigree	Pedigree to get size of
generation_upper_bound_in_result	Limit on generation to include in count; -1 means no limit. 0 only include the final generation. 1 only second last generation etc.

plot.malan_pedigree	<i>Plot pedigree</i>
---------------------	----------------------

Description

Plot malan_pedigree.

Usage

```
## S3 method for class 'malan_pedigree'
plot(
  x,
  ids = TRUE,
  haplotypes = FALSE,
  locus_sep = " ",
  mark_pids = NULL,
  label_color = "black",
  node_color = "lightgray",
  mark_color = "orange",
  ...
)
```

Arguments

x	Pedigree
ids	Show pids
haplotypes	Show haplotypes
locus_sep	Locus separator in haplotypes
mark_pids	Vector of pids to highlight
label_color	Label color
node_color	Node color
mark_color	Highlight color
...	Passed to <code>igraph::plot.igraph</code>

plot.malan_pedigreelist

Plot pedigree list

Description

Plot malan_pedigreelist generated by [build_pedigrees\(\)](#).

Usage

```
## S3 method for class 'malan_pedigreelist'
plot(x, ...)
```

Arguments

x	Pedigree list
...	ignored

population_populate_autosomal_infinite_alleles

Populate 1-locus autosomal DNA profile in pedigrees with infinite alleles mutation model.

Description

Populate 1-locus autosomal DNA profile from founder and down in all pedigrees. Note, that all founders have type 0 to begin with.

Usage

```
population_populate_autosomal_infinite_alleles(
  population,
  mutation_rate,
  progress = TRUE
)
```

Arguments

population	Population in which to populate genotypes
mutation_rate	Mutation rate between 0 and 1 (both included)
progress	Show progress

Details

The maternal allele is taken by random from the $2 \times N[g]$ alleles in the previous generation consisting of $N[g]$ males with descendants in the live population.

This is also why this is not using pedigrees but instead the population.

Note, that pedigrees need not be inferred.

See Also

[pedigrees_all_populate_haplotypes_custom_founders\(\)](#) and [pedigrees_all_populate_haplotypes_ladder_boundaries\(\)](#)

population_size_generation

Size of population

Description

Get the size of the population. Note the `generation_upper_bound_in_result` parameter.

Usage

```
population_size_generation(population, generation_upper_bound_in_result = -1L)
```

Arguments

population Population to get size of
 generation_upper_bound_in_result
 Limit on generation to include in count; -1 means no limit. 0 only include the final generation. 1 only second last generation etc.

```
print.malan_pedigree    Print pedigree
```

Description

Print pedigree

Usage

```
## S3 method for class 'malan_pedigree'
print(x, ...)
```

Arguments

x Pedigree
 ... ignored

```
print.malan_pedigreelist  

                         Print pedigree list
```

Description

Print malan_pedigreelist generated by [build_pedigrees\(\)](#).

Usage

```
## S3 method for class 'malan_pedigreelist'
print(x, ...)
```

Arguments

x Pedigrees (malan_pedigreelist)
 ... ignored

```
print.malan_population
```

Print population

Description

Print malan_population generated by [sample_geneology\(\)](#) or [sample_geneology_varying_size\(\)](#).

Usage

```
## S3 method for class 'malan_population'
print(x, ...)
```

Arguments

x	Population (malan_population)
...	ignored

```
print.malan_population_abort
```

Print malan_population_abort

Description

Print malan_population_abort

Usage

```
## S3 method for class 'malan_population_abort'
print(x, ...)
```

Arguments

x	malan_population_abort
...	ignored

print_individual	<i>Print individual</i>
------------------	-------------------------

Description

Print individual

Usage

```
print_individual(individual)
```

Arguments

individual	Individual
------------	------------

Examples

```
sim <- sample_geneology(100, 10)
indv <- get_individual(sim$population, 1)
print_individual(indv)
```

relationship_allele_diff_dist	<i>Calculate distribution of allele difference</i>
-------------------------------	--

Description

Calculate distribution of allele difference after m meioses.

Usage

```
relationship_allele_diff_dist(meioses, mu_dw, mu_up, method = "explicit")
```

Arguments

meioses	number of meioses separating the two individuals
mu_dw	mutation rate for 1-step down-mutation
mu_up	mutation rate for 1-step up-mutation
method	"explicit" (default): use known formulas for eigenvalues and eigenvectors. Can cause numerical problems. "matmult": do matrix multiplication instead of diagonalisation. "matmult_mpfpr": as "matmult" but with the Rmpfr library (note that this returns list instead of data.frame). "r_eigen": use R's eigen() function to find eigen values. Mostly for debugging.

Value

data.frame with columns d (allele difference) and p (prob)

relationship_allele_diff_dist_sym

Calculate distribution of allele difference for symmetric mutation rates

Description

Calculate distribution of allele difference after m meioses.

Usage

```
relationship_allele_diff_dist_sym(meioses, mu_updw, method = "explicit")
```

Arguments

meioses	number of meioses separating the two individuals
mu_updw	mutation rate for 1-step down- and up-mutations, i.e. total mutation rate is 2*mu_updw
method	"explicit" (default): use known formulas for eigenvalues and eigenvectors. Can cause numerical problems. "matmult": do matrix multiplication instead of diagonalisation. "matmult_mpfpr": as "matmult" but with the Rmpfr library (note that this returns list instead of data.frame). "r_eigen": use R's eigen() function to find eigen values. Mostly for debugging.

Value

data.frame with columns d (allele difference) and p (prob)

sample_autosomal_genotype

Sample genotype with theta

Description

Sample genotype with theta

Usage

```
sample_autosomal_genotype(allele_dist, theta)
```

Arguments

allele_dist	Allele distribution (probabilities) – gets normalised
theta	Theta correction between 0 and 1 (both included)

sample_geneology	<i>Simulate a geneology with constant population size.</i>
------------------	--

Description

This function simulates a geneology where the last generation has population_size individuals.

Usage

```
sample_geneology(
  population_size,
  generations,
  generations_full = 1L,
  generations_return = 3L,
  enable_gamma_variance_extension = FALSE,
  gamma_parameter_shape = 5,
  gamma_parameter_scale = 1/5,
  progress = TRUE,
  verbose_result = FALSE
)
```

Arguments

population_size	The size of the population.
generations	The number of generations to simulate: • -1 for simulate to 1 founder • else simulate this number of generations.
generations_full	Number of full generations to be simulated.
generations_return	How many generations to return (pointers to) individuals for.
enable_gamma_variance_extension	Enable symmetric Dirichlet (and disable standard Wright-Fisher).
gamma_parameter_shape	Parameter related to symmetric Dirichlet distribution for each man's probability to be father. Refer to details.
gamma_parameter_scale	Parameter related to symmetric Dirichlet distribution for each man's probability to be father. Refer to details.
progress	Show progress.
verbose_result	Verbose result.

Details

By the backwards simulating process of the Wright-Fisher model, individuals with no descendants in the end population are not simulated. If for some reason additional full generations should be simulated, the number can be specified via the `generations_full` parameter. This can for example be useful if one wants to simulate the final 3 generations although some of these may not get (male) children.

Let α be the parameter of a symmetric Dirichlet distribution specifying each man's probability to be the father of an arbitrary male in the next generation. When $\alpha = 5$, a man's relative probability to be the father has 95\ constant 1 under the standard Wright-Fisher model and the standard deviation in the number of male offspring per man is 1.10 (standard Wright-Fisher = 1).

This symmetric Dirichlet distribution is implemented by drawing father (unscaled) probabilities from a Gamma distribution with parameters `gamma_parameter_shape` and `gamma_parameter_scale` that are then normalised to sum to 1. To obtain a symmetric Dirichlet distribution with parameter α , the following must be used: '`gamma_parameter_shape`' = α and '`gamma_parameter_scale`' = $1/\alpha$.

Value

A `malan_simulation` / list with the following entries:

- `population`. An external pointer to the population.
- `generations`. Generations actually simulated, mostly useful when parameter `generations` = -1.
- `founders`. Number of founders after the simulated generations.
- `growth_type`. Growth type model.
- `sdo_type`. Standard deviation in a man's number of male offspring. `StandardWF` or `GammaVariation` depending on `enable_gamma_variance_extension`.
- `end_generation_individuals`. Pointers to individuals in end generation.
- `individuals_generations`. Pointers to individuals in last `generations_return` generation (if `generations_return` = 3, then individuals in the last three generations are returned).

If `verbose_result` is true, then these additional components are also returned:

- `individual_pids`. A matrix with pid (person id) for each individual.
- `father_pids`. A matrix with pid (person id) for each individual's father.
- `father_indices`. A matrix with indices for fathers.

See Also

[sample_geneology_varying_size\(\)](#).

Examples

```
sim <- sample_geneology(100, 10)
str(sim, 1)
sim$population
peds <- build_pedigrees(sim$population)
```

peds

sample_geneology_varying_size

Simulate a geneology with varying population size.

Description

This function simulates a geneology with varying population size specified by a vector of population sizes, one for each generation.

Usage

```
sample_geneology_varying_size(
  population_sizes,
  generations_full = 1L,
  generations_return = 3L,
  enable_gamma_variance_extension = FALSE,
  gamma_parameter_shape = 5,
  gamma_parameter_scale = 1/5,
  progress = TRUE
)
```

Arguments

population_sizes

The size of the population at each generation, g. population_sizes[g] is the population size at generation g. The length of population_sizes is the number of generations being simulated.

generations_full

Number of full generations to be simulated.

generations_return

How many generations to return (pointers to) individuals for.

enable_gamma_variance_extension

Enable symmetric Dirichlet (and disable standard Wright-Fisher).

gamma_parameter_shape

Parameter related to symmetric Dirichlet distribution for each man's probability to be father. Refer to details.

gamma_parameter_scale

Parameter related to symmetric Dirichlet distribution for each man's probability to be father. Refer to details.

progress

Show progress.

Details

By the backwards simulating process of the Wright-Fisher model, individuals with no descendants in the end population are not simulated. If for some reason additional full generations should be simulated, the number can be specified via the `generations_full` parameter. This can for example be useful if one wants to simulate the final 3 generations although some of these may not get (male) children.

Let α be the parameter of a symmetric Dirichlet distribution specifying each man's probability to be the father of an arbitrary male in the next generation. When $\alpha = 5$, a man's relative probability to be the father has 95% constant 1 under the standard Wright-Fisher model and the standard deviation in the number of male offspring per man is 1.10 (standard Wright-Fisher = 1).

This symmetric Dirichlet distribution is implemented by drawing father (unscaled) probabilities from a Gamma distribution with parameters `gamma_parameter_shape` and `gamma_parameter_scale` that are then normalised to sum to 1. To obtain a symmetric Dirichlet distribution with parameter α , the following must be used: '`gamma_parameter_shape`' = α and '`gamma_parameter_scale`' = $1/\alpha$.

Value

A `malan_simulation` / list with the following entries:

- `population`. An external pointer to the population.
- `generations`. Generations actually simulated, mostly useful when parameter `generations` = -1.
- `founders`. Number of founders after the simulated generations.
- `growth_type`. Growth type model.
- `sdo_type`. Standard deviation in a man's number of male offspring. `StandardWF` or `GammaVariation` depending on `enable_gamma_variance_extension`.
- `end_generation_individuals`. Pointers to individuals in end generation.
- `individuals_generations`. Pointers to individuals in last `generations_return` generation (if `generations_return` = 3, then individuals in the last three generations are returned).

See Also

[sample_geneology\(\)](#).

Examples

```
sim <- sample_geneology_varying_size(10*(1:10))
str(sim, 1)
sim$population
peds <- build_pedigrees(sim$population)
peds
```

set_generation	<i>Set individual's generation number</i>
----------------	---

Description

Note that generation 0 is final, end generation. 1 is second last generation etc.

Usage

```
set_generation(individual, generation)
```

Arguments

individual	Individual
generation	Generation to assign

Examples

```
sim <- sample_geneology(100, 10)
indv <- get_individual(sim$population, 1)
get_generation(indv)
set_generation(indv, 100)
get_generation(indv)
```

split_by_haplotypes	<i>Split pids by haplotype</i>
---------------------	--------------------------------

Description

Individuals with the same haplotype will be in the same group and individuals with different haplotypes will be in different groups.

Usage

```
split_by_haplotypes(population, pids)
```

Arguments

population	Population obtained from simulation
pids	Vector of individual pids

Value

List of integer vector, element *i* is an IntegerVector with all pids from pids with the same haplotype

test_create_population	<i>Generate test population</i>
------------------------	---------------------------------

Description

Generate test population

Usage

```
test_create_population()
```

Value

An external pointer to the population.

ystr_kits	<i>Kit information about Y-STR markers</i>
-----------	--

Description

A dataset containing information about the Y chromosomal short tandem repeat (Y-STR) markers that are present in the kit.

Usage

```
ystr_kits
```

Format

A data frame with 88 rows and 2 variables:

Marker name of Y-STR marker

Kit name of Y-STR kit

Source

<https://www.yhrd.org>

ystr_markers

Mutational information about Y-STR markers

Description

A dataset from yhrd.org (and their sources) containing mutational information about Y chromosomal short tandem repeat (Y-STR) markers used in forensic genetics.

Usage

```
ystr_markers
```

Format

A data frame with 29 rows and 5 variables:

Marker name of Y-STR marker

Meioses number of meioses observed

Mutations number of mutations observed in the corresponding number of Meioses

MutProb point estimate of mutation probability, MutProb = Mutations/Meioses

Alleles observed alleles

Details

Note, that loci with duplications (DYS385a/b as well as DYF387S1a/b have been split into two loci).

Source

<https://www.yhrd.org>

[[.malan_pedigreelist *Get pedigree from pedigree list*

Description

Get pedigree from malan_pedigreelist generated by `build_pedigrees()`.

Usage

```
## S3 method for class 'malan_pedigreelist'
x[[]]
```

Arguments

x	Element id
...	ignored

Value

Pedigree

[[.malan_population	<i>Get individual from population by pid</i>
---------------------	--

Description

Get individual from population by pid

Usage

```
## S3 method for class 'malan_population'  
x[[...]]
```

Arguments

x	pid
...	ignored

Value

Individual

Index

* datasets

ystr_kits, [63](#)
ystr_markers, [64](#)
[[.malan_pedigreelist, [64](#)
[[.malan_population, [65](#)

analyse_mixture_result, [5](#)
analyse_mixture_result(), [5](#)
analyse_mixture_results, [5](#)
as_tbl_graph(), [29](#)
as_tbl_graph.malan_pedigreelist, [6](#)

brothers_matching, [7](#)
build_haplotype_hashmap, [7](#)
build_haplotype_hashmap(), [13](#), [28](#)
build_pedigrees, [8](#)
build_pedigrees(), [36](#), [38](#), [39](#), [43–45](#), [47](#),
[52](#), [54](#), [64](#)

calc_autosomal_genotype_conditional_cumdist,
[8](#)
calc_autosomal_genotype_probs, [9](#)
construct_M, [9](#)
count_brothers, [10](#)
count_haplotype_near_matches_individuals,
[10](#)
count_haplotype_near_matches_individuals(),
[11](#), [50](#)
count_haplotype_occurrences_individuals,
[11](#)
count_haplotype_occurrences_individuals(),
[11](#), [49](#)
count_haplotype_occurrences_pedigree,
[12](#)
count_uncles, [12](#)

delete_haplotypeids_hashmap, [13](#)

eigen(), [56](#), [57](#)
estimate_autotheta_1subpop_genotypes,
[13](#)

estimate_autotheta_1subpop_individuals,
[14](#)
estimate_autotheta_subpops_genotypes,
[15](#)
estimate_autotheta_subpops_individuals,
[16](#)
estimate_autotheta_subpops_pids, [16](#)
estimate_autotheta_subpops_unweighted_genotypes,
[17](#)
estimate_autotheta_subpops_unweighted_pids,
[18](#)

father_matches, [18](#)
from_igraph, [19](#)
from_igraph_rcpp, [19](#)

generate_get_founder_haplotype_db, [20](#)
generate_get_founder_haplotype_ladder,
[20](#)
get_allele_counts_genotypes, [21](#)
get_allele_counts_pids, [21](#)
get_brothers, [22](#)
get_brothers(), [10](#), [22–24](#), [33](#)
get_children, [22](#)
get_children(), [22–24](#), [33](#)
get_cousins, [23](#)
get_cousins(), [22](#), [24](#), [33](#)
get_family_info, [23](#)
get_father, [24](#)
get_father(), [22](#)
get_generation, [24](#)
get_haplotype, [25](#)
get_haplotypes_in_pedigree, [26](#)
get_haplotypes_individuals, [25](#)
get_haplotypes_individuals(), [25](#), [27](#)
get_haplotypes_pids, [27](#)
get_haplotypes_pids(), [25](#), [26](#)
get_individual, [27](#)
get_individuals, [28](#)
get_matching_pids_from_hashmap, [28](#)

- get_matching_pids_from_hashmap(), 7, 13
- get_nodes_edges, 29
- get_pedigree_as_graph, 30
- get_pedigree_from_individual, 30
- get_pedigree_id, 31
- get_pedigree_id_from_pid, 31
- get_pedigrees_tidy, 29
- get_pid, 32
- get_pids_in_pedigree, 32
- get_uncles, 33
- get_uncles(), 13, 22–24
- get_zero_haplotype_generator, 33
- grandfather_matches, 34

- haplotype_matches_individuals, 35
- haplotype_partially_matches_individuals, 35
- haplotypes_to_hashes, 34

- infer_generation, 36
- infer_generations, 36

- load_haplotypes, 37
- load_individuals, 37

- malan (malan-package), 4
- malan-package, 4
- meioses_generation_distribution, 38
- meiotic_dist, 38
- meiotic_dist_threshold, 39
- meiotic_radius, 39
- mixture_info_by_individuals_2pers, 40, 41, 42
- mixture_info_by_individuals_2pers(), 5, 6
- mixture_info_by_individuals_3pers, 40, 40, 41, 42
- mixture_info_by_individuals_3pers(), 5, 6
- mixture_info_by_individuals_4pers, 40, 41, 41, 42
- mixture_info_by_individuals_4pers(), 5, 6
- mixture_info_by_individuals_5pers, 40, 41, 42
- mixture_info_by_individuals_5pers(), 5, 6

- pedigree_as_igraph, 48

- pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists, 49
- pedigree_haplotype_matches_in_pedigree_meiosis_L1_dists(), 11, 12, 35
- pedigree_haplotype_near_matches_meiosis, 50
- pedigree_size, 50
- pedigree_size_generation, 51
- pedigrees_all_populate_autosomal, 43
- pedigrees_all_populate_autosomal(), 14–18, 21
- pedigrees_all_populate_haplotypes, 44
- pedigrees_all_populate_haplotypes(), 25, 27, 45, 47
- pedigrees_all_populate_haplotypes_custom_founders, 45
- pedigrees_all_populate_haplotypes_custom_founders(), 25, 27, 43, 44, 47, 53
- pedigrees_all_populate_haplotypes_ladder_bounded, 46
- pedigrees_all_populate_haplotypes_ladder_bounded(), 25, 27, 43–45, 53
- pedigrees_count, 47
- pedigrees_table, 48
- plot.malan_pedigree, 51
- plot.malan_pedigreelist, 52
- population_populate_autosomal_infinite_alleles, 53
- population_size_generation, 53
- print.malan_pedigree, 54
- print.malan_pedigreelist, 54
- print.malan_population, 55
- print.malan_population_abort, 55
- print_individual, 56

- relationship_allele_diff_dist, 56
- relationship_allele_diff_dist_sym, 57

- sample_autosomal_genotype, 57
- sample_geneology, 58
- sample_geneology(), 8, 55, 61
- sample_geneology_varying_size, 60
- sample_geneology_varying_size(), 8, 55, 59
- set_generation, 62
- split_by_haplotypes, 62

- tbl_graph(), 6
- test_create_population, 63

ystr_kits, [63](#)

ystr_markers, [64](#)