Package ‘purgeR’

December 7, 2022

Type Package

Title Inbreeding-Purging Estimation in Pedigreed Populations

Version 1.7

Date 2022-12-07

Description Inbreeding-purging analysis of pedigreed populations, including the computation of the inbreeding coefficient, partial, ancestral and purged inbreeding coefficients, and measures of the opportunity of purging related to the individual reduction of inbreeding load. In addition, functions to calculate the effective population size and other parameters relevant to population genetics are included. See López-Cortegano E. (2021) <doi:10.1093/bioinformatics/btab599>.

URL https://gitlab.com/elcortegano/purgeR/

BugReports https://gitlab.com/elcortegano/purgeR/-/issues/

Encoding UTF-8

License GPL-2

Depends R (>= 3.5.0)

Imports doSNOW (>= 1.0.19), foreach (>= 1.5.1), parallel, progress (>= 1.2.2), Rcpp (>= 1.0.5), RcppProgress (>= 0.4.2)

LinkingTo Rcpp, RcppProgress

Suggests caret (>= 6.0-86), coda (>= 0.19-4), dplyr (>= 1.0.0), e1071 (>= 1.7-4), ggplot2 (>= 3.3.1), glmnet (>= 4.0-2), gtable (>= 0.3.0), knitr (>= 1.28), magrittr (>= 1.5), pander (>= 0.6.3), plyr (>= 1.8.6), purrr (>= 0.3.4), rmarkdown (>= 2.2), scales (>= 1.1.1), stringr (>= 1.4.0), testthat (>= 2.3.2), tibble (>= 3.0.1), tidy (>= 1.1.0), tidyselect (>= 1.1.0)

RoxygenNote 7.2.0

VignetteBuilder knitr, rmarkdown

LazyData true

NeedsCompilation yes

Author Eugenio López-Cortegano [aut, cre] (<https://orcid.org/0000-0001-6914-6305>)
R topics documented:

ancestors ........................................ 3
arrui ................................................. 4
atlas .................................................. 5
check_ancestors ................................... 6
check_basic ......................................... 6
check_bool .......................................... 7
check_col ........................................... 8
check_d ............................................... 8
check_df ............................................. 9
check_Fcol .......................................... 9
check_index ........................................ 10
check_int .......................................... 10
check_length ...................................... 11
check_na .......................................... 11
check_names ...................................... 12
check_Ne .......................................... 12
check_not_col .................................... 13
check_nrows ...................................... 13
check_order ...................................... 14
check_reference .................................. 14
check_repeat_id .................................. 15
check_tcol ....................................... 15
check_types ...................................... 16
check_zero_id ................................... 16
dama .................................................. 17
darwin .............................................. 18
delta_Fi ........................................... 18
dorcas .............................................. 19
exp_F .............................................. 20
exp_Fa ............................................ 21
exp_g ................................................ 22
F .................................................... 23
Fa ................................................... 23
Fij_core .......................................... 24
Fij_core_i_cpp .................................... 25
g ..................................................... 25
hwde ............................................... 26
idx_ancestors .................................... 27
ip_F ................................................ 27
ip_Fa ............................................. 28
ip_Fij ............................................. 29
Description

Returns a boolean vector indicating what individuals are suitable for purging analyses, given a measure of fitness. Individuals with NA values of fitness, and that do not have descendants with non-NA fitness values, are excluded.

Usage

ancestors(ped, reference, rp_idx, nboot = 10000L, seed = NULL, skip_Ng = FALSE)

Arguments

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **reference**: A string naming a column indicating whether individuals belong to the reference population or not. Column must be boolean or coercible to boolean type.
- **rp_idx**: Vector containing the indexes of individuals of the RP
- **nboot**: Number of bootstrap iterations (for computing Ng).
- **seed**: Sets a seed for the random number generator.
- **skip_Ng**: Skip Ng computation or not (FALSE by default).
Value

Boolean vector indicating what individuals will be evaluated.

---

Arrui pedigree

Description

This data set contains the pedigree of the arrui (*Ammotragus lervia*), also known as barbary sheep. A total of 380 individuals is included, as well as measurements of biological fitness and other factors (see reference below for details).

Usage

arrui

Format

A data frame with with records from 380 individuals (in rows), and 10 variables:

- *id* - Individual identity.
- *dam* - Maternal identity.
- *sire* - Paternal identity.
- *survival15* - 15-days survival.
- *prod* - Female productivity.
- *sex* - Individual sex.
- *yob* - Year of birth.
- *pom* - Period of management.
- *target* - Individual in the target population.
- *eeza_id* - Individual identity (as recorded in the original studbook)

Source

The original studbook containing the complete and updated pedigree can be found at: http://www.eeza.csic.es/en/programadecria.aspx.

References

Description

This data set contains the pedigree of Cuvier’s gazelle (*Gazella atlas*). A total of 948 individuals is included, as well as measurements of biological fitness and other factors (see reference below for details).

Usage

atlas

Format

A data frame with with records from 948 individuals (in rows), and 10 variables:

- *id* - Individual identity.
- *dam* - Maternal identity.
- *sire* - Paternal identity.
- *survival15* - 15-days survival.
- *prod* - Female productivity.
- *sex* - Individual sex.
- *yob* - Year of birth.
- *pom* - Period of management.
- *target* - Individual in the target population.
- *eeza_id* - Individual identity (as recorded in the original studbook)

Source

The original studbook containing the complete and updated pedigree can be found at: http://www.eeza.csic.es/en/programadecria.aspx.

References

check_ancestors  Check ancestor individuals

Description

Takes a column name, and checks its use as target. It should name a boolean vector (or coercible to
it), with at least one TRUE value.

Usage

check_ancestors(id, ancestors)

Arguments

id  Vector of individual ids.
ancestors  Vector of ancestor ids.

Value

No return value. Will print an error message if checking fail.

check_basic  Check basic

Description

This function will group some other checking functions, that should be run everytime when using
functions in this package, to avoid unexpected errors.

Usage

check_basic(
  ped,
  id_name = "id",
  dam_name = "dam",
  sire_name = "sire",
  when_rename = FALSE,
  when_sort = FALSE
)

check_bool

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ped</td>
<td>A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.</td>
</tr>
<tr>
<td>id_name</td>
<td>Column name for individual id.</td>
</tr>
<tr>
<td>dam_name</td>
<td>Column name for dam.</td>
</tr>
<tr>
<td>sire_name</td>
<td>Column name for sire.</td>
</tr>
<tr>
<td>when_rename</td>
<td>True when called from ped_rename function. It softs checks on individual ID column name and types</td>
</tr>
<tr>
<td>when_sort</td>
<td>True when called from ped_sort function. It softs checks on pedigree sorting</td>
</tr>
</tbody>
</table>

Value

No return value. Will print an error message if checking fail.

---

check_bool [Check if a variable is boolean or not]

Description

Can be used to test arguments that need to be of logical (boolean) type

Usage

check_bool(variable)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>Variable to test</td>
</tr>
</tbody>
</table>

Value

No return value. Will print an error message if checking fail.
check_col

Check that optional column is included

Description
Some functions require additional columns. Check that they are named in the pedigree.

Usage
check_col(names, name)

Arguments
- names: Column names (all)
- name: Column name to check.

Value
No return value. Will print an error message if checking fail.

check_d

Check purging coefficient

Description
The purging coefficient must be a number between 0 and 0.5

Usage
check_d(d)

Arguments
- d: Purging coefficient (taking values between 0.0 and 0.5).

Value
No return value. Will print an error message if checking fail.
**check_df**

*Check pedigree class*

**Description**

The pedigree must be of object class 'data.frame'.

**Usage**

```r
check_df(obj)
```

**Arguments**

- **obj**  
  Object to test

**Value**

No return value. Will print an error message if checking fail.

---

**check_Fcol**

*Check columns with inbreeding values*

**Description**

Takes a column name, and checks its use as inbreeding coefficient. It should name a numeric vector, with values in the range [0,1]

**Usage**

```r
check_Fcol(ped, Fcol, compute = TRUE)
```

**Arguments**

- **ped**  
  A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **Fcol**  
  Name of column with inbreeding coefficient values. If none is used, inbreeding will be computed.
- **compute**  
  Compute inbreeding if Fcol is NULL

**Value**

Vector of inbreeding values (if checks are successful)
check_index  

Check individual index

Description
Renamed individuals must be named by their index (from 1 to N)

Usage
check_index(id)

Arguments
id  Column of individual ids.

Value
No return value. Will print an error message if checking fail.

check_int  Check if a variable is a positive integer or not

Description
Can be used to test arguments that need to be integers

Usage
check_int(variable)

Arguments
variable  Variable to test

Value
No return value. Will print an error message if checking fail.
**check_length**

Check if a variable has length >1

**Description**

Used to test arguments that need to be of length 1

**Usage**

```r
check_length(variable, message = "Expected value of length 1")
```

**Arguments**

- variable: Variable to test
- message: Error message to display

**Value**

No return value. Will print an error message if checking fail.

---

**check_na**

Check if a vector contains NA values

**Description**

Return warning when NA values are present

**Usage**

```r
check_na(variable)
```

**Arguments**

- variable: Variable to test

**Value**

No return value. Will print an error message if checking fail.
check_names

*Check that mandatory column names are included*

**Description**

Columns for id, dam and sire are mandatory. This function checks that they are named in the pedigree. The function works with arbitrary column names (not 'id', 'dam' and 'sire') to work with `ped_rename()`.

**Usage**

```r
cHECK_names(ped, id_name = "id", dam_name = "dam", sire_name = "sire")
```

**Arguments**

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **id_name**: Column name for individual id.
- **dam_name**: Column name for dam.
- **sire_name**: Column name for sire.

**Value**

No return value. Will print an error message if checking fail.

check_Ne

*Check Ne*

**Description**

The effective population size (Ne) must be a number higher than 0.

**Usage**

```r
cHECK_Ne(Ne)
```

**Arguments**

- **Ne**: Effective population size.

**Value**

No return value. Will print an error message if checking fail.
**check_not_col**

Check if optional column is included

**Description**
Some functions require additional columns. Check if they are already named in the pedigree.

**Usage**
```
check_not_col(names, name)
```

**Arguments**
- `names` Column names (all)
- `name` Column name to check.

**Value**
No return value. Will print an error message if checking fail.

**check_nrows**

Check observed and expected number of rows

**Description**
Expected and observed number of rows must be equal.

**Usage**
```
check_nrows(df, exp, message = "Expected value of length 1")
```

**Arguments**
- `df` Dataframe to test
- `exp` Expected number of rows
- `message` Error message to display

**Value**
No return value. Will print an error message if checking fail.
check_order  

**Check individual order**

**Description**

Individuals must be sorted from older to younger

**Usage**

```r
cHECK_ORDER(id, dam, sire, soft_sorting = FALSE)
```

**Arguments**

- `id` Vector of individual ids.
- `dam` Vector of dam ids.
- `sire` Vector of sire ids.
- `soft_sorting` If TRUE checking is relaxed, allowing descendants to be declared before ancestors

**Value**

No return value. Will print an error message if checking fail.

---

check_reference  

**Check columns with reference individuals**

**Description**

Takes a column name, and checks its use as reference. It should name a boolean vector (or coercible to it), with at least one TRUE value.

Takes a column name, and checks its use as target. It should name a boolean vector (or coercible to it), with at least one TRUE value.

**Usage**

```r
cHECK_REFERENCE(ped, reference)
cHECK_TARGET(ped, reference, target, variable)
```

**Arguments**

- `ped` A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- `reference` A string naming a column indicating whether individuals belong to the reference population or not. Column must be boolean or coercible to boolean type.
- `target` Target column
- `variable` To be used in printed messages
check_repeat_id

Value
Vector of reference numbers (if checks are successful)
Vector of target numbers (if checks are successful)

Description
Individual id are unique and cannot be repeated

Usage
check_repeat_id(id)

Arguments
id Vector of individual ids.

Value
No return value. Will print an error message if checking fail.

check_tcol Check columns with generation numbers

Description
Takes a column name, and checks its use as generation numbers. It should name a numeric vector, with values >= 0.

Usage
check_tcol(ped, tcol, compute = TRUE, force_int = FALSE)

Arguments
ped A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
tcol Name of column with individual generation times. If none is used, the number of equivalent complete generations is computed.
compute Compute generation numbers if tcol is NULL
force_int Generation numbers must be integers (disabled by default)

Value
Vector of generation numbers (if checks are successful)
**check_types**

*Check that mandatory column names are of type int*

**Description**

Columns for id, dam and sire are mandatory, and required to be of type integer.

**Usage**

```r
cHECK_TYPES(id, dam, sire)
```

**Arguments**

- **id**: Vector of individual ids.
- **dam**: Vector of dam ids.
- **sire**: Vector of sire ids.

**Value**

No return value. Will print an error message if checking fail.

---

**check_zero_id**

*Check individuals named zero*

**Description**

Individual id cannot equal zero (0). This is reserved to dams and sires.

**Usage**

```r
cHECK_ZERO_ID(id)
```

**Arguments**

- **id**: Vector of individual ids.

**Value**

No return value. Will print an error message if checking fail.
Description

This data set contains the pedigree of the dama gazelle (*Nanger dama*). A total of 1316 individuals is included, as well as measurements of biological fitness and other factors (see reference below for details).

Usage

dama

Format

A data frame with with records from 1316 individuals (in rows), and 10 variables:

- *id* - Individual identity.
- *dam* - Maternal identity.
- *sire* - Paternal identity.
- *survival15* - 15-days survival.
- *prod* - Female productivity.
- *sex* - Individual sex.
- *yob* - Year of birth.
- *pom* - Period of management.
- *target* - Individual in the target population.
- *eeza_id* - Individual identity (as recorded in the original studbook)

Source

The original studbook containing the complete and updated pedigree can be found at: http://www.eeza.csic.es/en/programadecria.aspx.

References

### darwin

**Darwin/Wedgwood pedigree**

**Description**

This data set contains the pedigree of the Darwin/Wedgwood dynasty. It is composed by a total of 63 individuals, including Charles R. Darwin and Francis Galton.

**Usage**

darwin

**Format**

A data frame with with records from 63 individuals (in rows), and 3 variables:

- *Individual* - Individual identity.
- *Mother* - Mother’s identity.
- *Father* - Father’s identity.

**Source**

The pedigree is adapted from Berra et al. (2010)

**References**


### delta_Fi

**Individual inbreeding variation**

**Description**

Computes the increase in inbreeding coefficient for a given individual

**Usage**

delta_Fi(Fi, t)

**Arguments**

- **Fi** Individual inbreeding coefficient.
- **t** Individual generation number.

**Value**

Individual variation in inbreeding.
**dorcas**  

---

**Dorcas gazelle pedigree**

---

**Description**

This data set contains the pedigree of dorcas gazelle (*Gazella dorcas*). A total of 1279 individuals is included, as well as measurements of biological fitness and other factors (see reference below for details).

**Usage**

dorcas

**Format**

A data frame with records from 1279 individuals (in rows), and 10 variables:

- **id** - Individual identity.
- **dam** - Maternal identity.
- **sire** - Paternal identity.
- **survival15** - 15-days survival.
- **prod** - Female productivity.
- **sex** - Individual sex.
- **yob** - Year of birth.
- **pom** - Period of management.
- **target** - Individual in the target population.
- **eeza_id** - Individual identity (as recorded in the original studbook)

**Source**

The original studbook containing the complete and updated pedigree can be found at: [http://www.eeza.csic.es/en/programadecria.aspx](http://www.eeza.csic.es/en/programadecria.aspx).

**References**

exp_F

Expected inbreeding coefficient

Description

Estimates the expected inbreeding coefficient (F) as a function of the effective population size and generation number.

Usage

\( \text{exp\_F}(N_e, t) \)

Arguments

- \( N_e \) Effective population size
- \( t \) Generation number

Details

Computation of the inbreeding coefficient uses the classical formula:

\[ F(t) = 1 - (1 - 1/2N)^t \]

Value

The inbreeding coefficient

References


See Also

- \( \text{ip\_F} \)

Examples

\[
\begin{align*}
\text{exp\_F}(N_e = 50, t = 0) \\
\text{exp\_F}(N_e = 50, t = 50) \\
\text{exp\_F}(N_e = 10, t = 50)
\end{align*}
\]
exp_Fa

Expected ancestral inbreeding coefficient

Description

Estimates the expected ancestral inbreeding coefficient (Fa) as a function of the effective population size and generation number

Usage

exp_Fa(Ne, t)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ne</td>
<td>Effective population size</td>
</tr>
<tr>
<td>t</td>
<td>Generation number</td>
</tr>
</tbody>
</table>

Details

Computation of the ancestral inbreeding coefficient uses the adaptation from Ballou’s (1997) formula, as in López-Cortegano et al. (2018):

\[ Fa(t) = 1 - (1 - 1/2N) ^ (1/2 (t-1)t) \]

Value

The ancestral inbreeding coefficient

References


See Also

ip_Fa

Examples

exp_Fa(Ne = 50, t = 0)
exp_Fa(Ne = 50, t = 50)
exp_Fa(Ne = 10, t = 50)
exp_g

Expected purged inbreeding coefficient

Description

Estimates the expected purged inbreeding coefficient (g) as a function of the effective population size, generation number, and purging coefficient.

Usage

`exp_g(Ne, t, d)`

Arguments

- `Ne`: Effective population size
- `t`: Generation number
- `d`: Purging coefficient (taking values between 0.0 and 0.5).

Details

Computation of the purged inbreeding coefficient is calculated as in García-Dorado (2012):

\[ g(t) = \left( 1 - \frac{1}{2N} \right) g(t-1) + \frac{1}{2N} \right] \times \left[ 1 - 2d F(t-1) \right] \]

When convergence is reached, the asymptotic value \( g(a) \) is returned:

\[ g(a) = \frac{(1 - 2d)}{(1 + 2d (2N-1))} \]

Value

The purged inbreeding coefficient

References


See Also

- `ip_g`

Examples

- `exp_g(Ne = 50, t = 0, d = 0.15)`
- `exp_g(Ne = 50, t = 50, d = 0.15)`
- `exp_g(Ne = 10, t = 50, d = 0.15)`
**F**

*Inbreeding coefficient*

**Description**

Computes the standard inbreeding coefficient \((F)\). This is the probability that two alleles on a locus are identical by descent (Falconer and Mackay 1996, Wright 1922), calculated from the genealogical coancestry matrix (Malécot 1948).

**Usage**

\[
F(ped, \text{name}_\text{to})
\]

**Arguments**

- `ped` A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- `name_to` A string naming the new output column.

**Value**

The input dataframe, plus an additional column named "F" with individual inbreeding coefficient values.

**References**


**Fa**

*Ancestral inbreeding coefficient*

**Description**

Computes the ancestral inbreeding coefficient \((Fa)\). This is the probability that an allele has been in homozygosity in at least one ancestor (Ballou 1997). A genedrop approach is included to compute unbiased estimates of \(Fa\) (Baumung et al. 2015).

**Usage**

\[
Fa(ped, F_i, \text{name}_\text{to}, \text{genedrop} = 0L, \text{seed} = \text{NULL})
\]
Fij_core

**Arguments**

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **Fi**: Vector of inbreeding coefficient values.
- **name_to**: A string naming the new output column.
- **genedrop**: Number of genedrop iterations to run. If set to zero (as default), Ballou’s Fa is computed.
- **seed**: Sets a seed for the random number generator.

**Value**

The input dataframe, plus an additional column named “Fa” with individual ancestral inbreeding coefficient values.

**References**


---

Fij_core  
Partial inbreeding coefficient (core function)

**Description**

Computes partial inbreeding coefficients, Fi(j). A coefficient Fi(j) can be read as the probability of individual i being homozygous for alleles derived from ancestor j.

**Usage**

Fij_core(ped, ancestors, ancestors_idx, Fi, mapa, ncores = 1, genedrop, seed)

**Arguments**

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **ancestors**: Vector of the identities to be assumed as founder ancestors.
- **ancestors_idx**: Index of ancestors.
- **Fi**: Vector of inbreeding coefficients.
- **mapa**: Map of ancestors.
- **ncores**: Number of cores to use for parallel computing (default = 1).
- **genedrop**: Enable genedrop simulation.
- **seed**: Sets a seed for the random number generator.
Value

A matrix of partial inbreeding coefficients. Fi(j) values can thus be read from row i and column j.

---

**Fij_core_i_cpp**

*Partial inbreeding coefficient (core function)*

Description

Computes partial inbreeding coefficients, Fi(j). A coefficient Fi(j) can be read as the probability of individual i being homozygous for alleles derived from ancestor j.

Usage

Fij_core_i_cpp(dam, sire, anc_idx, mapa, Fi, genedrop = 0L, seed = NULL)

Arguments

- **dam**: Vector of dam ids.
- **sire**: Vector of sire ids.
- **anc_idx**: Index of ancestors.
- **mapa**: Map of ancestors.
- **Fi**: Vector of inbreeding coefficients.
- **gendrop**: Enable genedrop simulation
- **seed**: Sets a seed for the random number generator.

Value

A matrix of partial inbreeding coefficients. Fi(j) values can thus be read from row i and column j.

---

**g**

*Purged inbreeding coefficient*

Description

Computes the purged inbreeding coefficient (g). This is the probability that two alleles on a locus are identical by descent, but relative to deleterious recessive alleles (García-Dorado 2012). The reduction in g relative to standard inbreeding (F) is given by an effective purging coefficient (d), that measures the strength of the deleterious recessive component in the genome. The coefficient g is computed following the methods for pedigrees in García-Dorado (2012) and García-Dorado et al. (2016).

Usage

g(ped, d, Fi, name_to)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ped</code></td>
<td>A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.</td>
</tr>
<tr>
<td><code>d</code></td>
<td>Purging coefficient (taking values between 0.0 and 0.5).</td>
</tr>
<tr>
<td><code>Fi</code></td>
<td>Vector of inbreeding coefficient values</td>
</tr>
<tr>
<td><code>name_to</code></td>
<td>A string naming the new output column.</td>
</tr>
</tbody>
</table>

Value

The input dataframe, plus an additional column named "g" followed by the purging coefficient, containing purged inbreeding coefficient values.

References


hwd

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwd</td>
<td>Deviation from Hardy-Weinberg equilibrium</td>
</tr>
</tbody>
</table>

Description

Computes the deviation from Hardy-Weinberg equilibrium following Caballero and Toro (2000).

Usage

hwd(ped, reference = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ped</code></td>
<td>A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.</td>
</tr>
<tr>
<td><code>reference</code></td>
<td>A string naming a column indicating whether individuals belong to the reference population or not. Column must be boolean or coercible to boolean type.</td>
</tr>
</tbody>
</table>

Value

A numeric value indicating the deviation from Hardy-Weinberg equilibrium.

References

### idx_ancestors

**Index ancestors**

**Description**

Creates a vector of length N (the number of individuals) Only coordinates for valid ancestors will be given.

**Usage**

```r
idx_ancestors(ids, N)
```

**Arguments**

- `ids` Ancestor identities
- `N` Total number of individuals

**Value**

A logical matrix.

### ip_F

**Inbreeding coefficient**

**Description**

Computes the standard inbreeding coefficient ($F$). This is the probability that two alleles on a locus are identical by descent (Falconer and Mackay 1996, Wright 1922), calculated from the genealogical coancestry matrix (Malécot 1948).

**Usage**

```r
ip_F(ped, name_to = "Fi")
```

**Arguments**

- `ped` A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- `name_to` A string naming the new output column.
The input dataframe, plus an additional column with individual inbreeding coefficient values (named "Fi" by default).

**References**

**See Also**
- exp_F

**Examples**
```r
data(dama)
dama <- ip_F(dama)
tail(dama)
```

**Description**
Computes the ancestral inbreeding coefficient ($Fa$). This is the probability that an allele has been in homozygosity in at least one ancestor (Ballou 1997). A genedrop approach is included to compute unbiased estimates of $Fa$ (Baumung et al. 2015).

**Usage**
```r
ip_Fa(ped, name_to = "Fa", genedrop = 0, seed = NULL, Fcol = NULL)
```

**Arguments**
- **ped** A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **name_to** A string naming the new output column.
- **genedrop** Number of genedrop iterations to run. If set to zero (as default), Ballou’s Fa is computed.
- **seed** Sets a seed for the random number generator.
- **Fcol** Name of column with inbreeding coefficient values. If none is used, inbreeding will be computed.
Value

The input dataframe, plus an additional column with individual ancestral inbreeding coefficient values (named "Fa" by default).

References


See Also

ip_F, exp_Fa

Examples

data(dama)
# dama <- ip_Fa(dama) # Compute F on the go (won't be kept in the pedigree).
dama <- ip_F(dama)
dama <- ip_Fa(dama, Fcol = 'Fi') # If F is computed in advance.
tail(dama)

Description

Computes partial inbreeding coefficients, \(F_i(j)\). A coefficient \(F_i(j)\) can be read as the probability of individual \(i\) being homozygous for alleles derived from ancestor \(j\). It is calculated following the tabular method described by Gulisija & Crow (2007). Optionally, it can be estimated via genedrop simulation.

Usage

\[
ip_Fij(
  ped,
  mode = "founders",
  ancestors = NULL,
  Fcol = NULL,
  genedrop = 0,
  seed = NULL,
  ncores = 1L
)
\]
Arguments

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.

- **mode**: Defines the set of ancestors considered when computing partial inbreeding. It can be set as: "founder" for inbreeding conditional to founders only (default), "all" for all individuals in the pedigree (it may take long to compute in large pedigrees), and "custom" for individuals identities given in a integer vector (see ancestors’ argument).

- **ancestors**: Under the "custom" run mode, it defines a vector of ancestors that will be considered when computing partial inbreeding values.

- **Fcol**: Name of column with inbreeding coefficient values. If none is used, inbreeding will be computed.

- **genedrop**: Number of genedrop iterations to run. If set to zero (as default), exact coefficients are computed.

- **seed**: Sets a seed for the random number generator (only if genedrop is enabled).

- **ncores**: Number of cores to use for parallel computing (default = 1)

Value

A matrix of partial inbreeding coefficients. $F_i(j)$ values can thus be read from row i and column j. In the resultant matrix, there are as many rows as individuals in the pedigree, and as many columns as ancestors used. Columns will be named and sorted by ancestor identity.

References


See Also

- `ip_F`

Examples

```r
# Original pedigree file in Gulisija & Crow (2007)
pedigree <- tibble::tibble(
  id = c("M", "K", "J", "a", "c", "b", "e", "d", "I"),
  dam = c("0", "0", "0", "K", "M", "a", "c", "c", "e"),
  sire = c("0", "0", "0", "J", "a", "J", "b", "b", "d")
)
pedigree <- purgeR::ped_rename(pedigree, keep_names = TRUE)

# Partial inbreeding relative to founder ancestors
m <- ip_Fij(pedigree)
# Note that in the example above, the sum of the values in
# rows will equal the vector of inbreeding coefficients
# i.e. base::rowSums(m) equals purgeR::ip_F(pedigree)$Fi

# Compute partial inbreeding relative to an arbitrary ancestor
```
### Description

Computes the purged inbreeding coefficient ($g$). This is the probability that two alleles on a locus are identical by descent, but relative to deleterious recessive alleles (García-Dorado 2012). The reduction in $g$ relative to standard inbreeding ($F$) is given by an effective purging coefficient ($d$), that measures the strength of the deleterious recessive component in the genome. The coefficient $g$ is computed following the methods for pedigrees in García-Dorado (2012) and García-Dorado et al. (2016).

### Usage

```r
ip_g(ped, d, name_to = "g<d>"; Fcol = NULL)
```

### Arguments

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **d**: Purging coefficient (taking values between 0.0 and 0.5).
- **name_to**: A string naming the new output column.
- **Fcol**: Name of column with inbreeding coefficient values. If none is used, inbreeding will be computed.

### Value

The input dataframe, plus an additional column containing purged inbreeding coefficient values (named "g" and followed by the purging coefficient value by default).

### References


### See Also

`ip_F`, `exp_g`
Examples

data(dama)
dama <- ip_g(dama, d = 0.23)
tail(dama)

Description

The potential reduction in individual inbreeding load can be estimated by means of the opportunity of purging (\(O\)) and expressed opportunity of purging (\(Oe\)) parameters described by Gulisija and Crow (2007). Whereas \(O\) relates to the total potential reduction of the inbreeding load in an individual, as a consequence of it having inbred ancestors, \(Oe\) relates to the expressed potential reduction of the inbreeding load. Only \(Oe\) is computed by default. Estimates of \(O\) and \(Oe\) need to be corrected in complex pedigrees (see Details below). Both corrected (named "O" and "Oe" by default), and non-corrected (suffixed with ".raw") are returned.

Usage

```r
ip_op(
  ped,
  name_Oe = "Oe",
  compute_O = FALSE,
  name_O = "O",
  Fcol = NULL,
  ncores = 1L,
  genedrop = 0,
  seed = NULL,
  complex = NULL
)
```

Arguments

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **name_Oe**: A string naming the new output column for the expressed opportunity of purging (defaults to "Oe")
- **compute_O**: Enable computation of total opportunity of purging (disabled by default)
- **name_O**: A string naming the new output column for total opportunity of purging (defaults to "O")
- **Fcol**: Name of column with inbreeding coefficient values. If none is used, inbreeding will be computed.
- **ncores**: Number of cores to use for parallel computing (default = 1)
gene drop Number of gene drop iterations run to compute partial inbreeding. If set to zero (as default), exact coefficients are computed.

seed Sets a seed for the random number generator (only if gene drop is enabled).

complex Enable correction for complex pedigrees (deprecated in v1.3, both raw and corrected measures of "Oe" are returned now).

Details
Model used here assume fully recessive, high effect size alleles (Gulisija and Crow, 2007).

In simple pedigrees, the opportunity of purging (O) and the expressed opportunity of purging (Oe) are estimated as in Gulisija and Crow (2007). For complex pedigrees involving more than one autozygous individual per path from an individual to an ancestor, O and Oe in the closer ancestors need to be discounted for what was already accounted for in their predecessors. To solve this problem, Gulisija and Crow (2007) provide expression to correct O and Oe (see equations 21 and 22 in the manuscript).

Here, an heuristic approach is used to prevent the inflation of O and Oe, and avoid the use of additional looped expressions that may result in an excessive computational cost. To do so, only the contribution of the most recent ancestors in a path will be considered. Specifically, the method skips contributions from "far" ancestors k, such that Fj(k) > 0, where j is an intermediate ancestor, both referred to an individual i of interest. Fj(k) refers to the partial inbreeding of j for alleles derived from k (see ip_Fij). This may not provide exact values of O and Oe, but we expect little bias, since more distant ancestors also contribute lesser to O and Oe.

Both types of estimates (corrected and non-corrected) are returned (non-corrected estimates, prefixed with "_raw").

Value
The input dataframe, plus an additional column containing Oe and Oe_raw estimates (additional columns for O can appended by enabling compute_O = TRUE).

References

See Also
ip_Fij

Examples
# Original pedigree file in Gulisija & Crow (2007)
pedigree <- tibble::tibble(
  id = c("M", "K", "J", "a", "c", "b", "e", "d", "I"),
  dam = c("0", "0", "0", "K", "M", "a", "c", "c", "e"),
  sire = c("0", "0", "0", "K", "M", "a", "c", "c", "e")
)
pedigree <- purgeR::ped_rename(pedigree, keep_names = TRUE)
ip_op(pedigree, compute_O = TRUE)
map_ancestors  

**Map ancestors**

**Description**

Creates a logical matrix that indicates whether an individual i (in columns) is ancestor of other j (in rows) For example, matrix[, 1] will indicate descendants of id = 1 And matrix[1, ] indicates ancestors of id = 1

**Usage**

map_ancestors(ped, idx)

**Arguments**

ped  
A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.

idx  
Index of ancestors to map

**Value**

A logical matrix.

---

Ne_delta  

**Realized effective population size (mean)**

**Description**

Computes the mean realized effective population size. Note this function expected a mean delta_F value for all individuals in the reference population

Computes the standard error of the realized effective population size. Note this function expects the mean and standard deviation of delta F, as well as the total number of individuals in the reference population

**Usage**

Ne_delta(delta)

se_Ne_delta(delta)

**Arguments**

delta  
Vector of individual variations in inbreeding.
Value

Mean effective population size.
Standard error of the effective population size.

---

**op**

*Opportunity of purging*

Description

The potential reduction in individual inbreeding load can be estimated by means of the opportunity of purging \((O)\) and expressed opportunity of purging \((Oe)\) parameters described by Gulisija and Crow (2007). Whereas \(O\) relates to the total potential reduction of the inbreeding load in an individual, as a consequence of it having inbred ancestors, \(Oe\) relates to the expressed potential reduction of the inbreeding load. In both cases, these measures are referred to fully recessive, high effect size alleles (e.g. lethals). For complex pedigrees, involving more than one autozygous individual per path from a reference individual to an ancestor, these estimates are estimated following an heuristic approach (see details below).

Usage

\[
op\(\text{ped, pi, Fi, name}_0, \text{name}_{0e}, \text{sufix, compute}_0 = \text{FALSE}\)
\]

Arguments

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **pi**: Partial inbreeding matrix
- **Fi**: Vector of inbreeding coefficient values
- **name_0**: A string naming the new output column for total opportunity of purging (defaults to "O")
- **name_{0e}**: A string naming the new output column for the expressed opportunity of purging (defaults to "Oe")
- **sufix**: A string naming the suffix for non-corrected O and Oe measures
- **compute_0**: Enable computation of total opportunity of purging (false by default)

Details

In simple pedigrees, the opportunity of purging \((O)\) and the expressed opportunity of purging \((Oe)\) are estimated as in Gulisija and Crow (2007). For complex pedigrees involving more than one autozygous individual per path from an individual to an ancestor, \(O\) and \(Oe\) in the closer ancestors need to be discounted for what was already accounted for in their predecessors. To solve this problem, Gulisija and Crow (2007) provide expression to correct \(O\) and \(Oe\) (see equations 21 and 22 in the manuscript).

Here, an heuristic approach is used to prevent the inflation of \(O\) and \(Oe\), and avoid the use of additional looped expressions that may result in an excessive computational cost. To do so, when
using `ip_op(complex = TRUE)` only the contribution of the most recent ancestors in a path will be considered. This may not provide exact values of $O$ and $Oe$, but we expect little bias, since more distant ancestors also contribute lesser to $O$ and $Oe$.

### Value

The input dataframe, plus two additional columns named "O" and "Oe", containing total and expressed opportunity of purging measures.

### References


---

**ped_clean**

Remove individuals not used for purging analyses

---

**Description**

Remove individuals that are not necessary for purging analyses involving fitness. This will reduce the size of the pedigree, and speed up the computation of inbreeding parameters. Individuals removed include those with unknown (NA) values of a given parameter, as long as they do not have any descendant in the pedigree with known values of that parameter. Cleaned pedigrees will automatically have individual identities renamed (see `ped_rename`).

**Usage**

```r
ped_clean(ped, value_from)
```

**Arguments**

- `ped` A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- `value_from` Name of the column of interest.

**Value**

A dataframe with the pedigree cleaned for the specified parameter (column) provided.

**See Also**

`ped_rename`

**Examples**

```r
data(arrui)
nrow(arrui)
arrui <- ped_clean(arrui, "survival15")
nrow(arrui)
```
ped_maternal  

**Description**

For every individual in the pedigree, it will assign them their maternal (or paternal) value for an observed variable of interest.

**Usage**

```r
ped_maternal(ped, value_from, name_to, use_dam = TRUE, set_na = NULL)
```

**Arguments**

- `ped`: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- `value_from`: Name of the column of interest.
- `name_to`: A string naming the new output column.
- `use_dam`: Extract maternal values. If false, parental values are returned.
- `set_na`: When maternal values are unknown, NA values are generated by default. This option allows to set a different value.

**Value**

The input dataframe, plus an additional column with maternal (or paternal) values of a variable of interest.

**Examples**

```r
# To assign maternal inbreeding as a new variable, we can do as follows:
data(dama)
dama <- ip_F(dama)
dama <- ped_maternal(dama, value_from = "Fi", name_to = "Fdam")
tail(dama)
```

---

ped_rename  

**Description**

Functions in `purgeR` require individuals to be named with integers from 1 to N. This takes a dataframe containing a pedigree, and rename individuals having names in any format to that required by other functions in `purgeR`. The process will also check that the pedigree format is suitable for other functions in the package.
**Usage**

```r
ped_rename(ped, id = "id", dam = "dam", sire = "sire", keep_names = FALSE)
```

**Arguments**

- `ped`: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- `id`: A string naming the column with individual identities. It will be renamed to its default value 'id'.
- `dam`: A string naming the column with maternal identities. It will be renamed to its default value 'dam'.
- `sire`: A string naming the column with paternal identities. It will be renamed to its default value 'sire'.
- `keep_names`: A boolean value indicating whether the original identity values should be kept on a separate column (named 'names'), or not.

**Value**

A dataframe with the pedigree’s identities renamed.

**See Also**

- `ped_clean`

**Examples**

```r
data(darwin)
darwin <- ped_rename(darwin, id = "Individual", dam = "Mother", sire = "Father", keep_names = TRUE)
head(darwin)
```

---

**ped_sort**

Sort individuals (with ancestors on top of descendants)

**Description**

Individuals can be sorted according to the pedigree structure, without need of birth dates. In the sorted pedigree, descendants will always be placed in rows with higher index number than that of their ancestors. This way, individuals born first will tend to be in the top of the pedigree. Younger individuals, and individuals with no descendants will tend to be placed at the bottom. This function uses the sorting algorithm developed by Zhang et al (2009). After sorting, individuals will be renamed from 1 to N using `ped_rename`.

**Usage**

```r
ped_sort(ped, id = "id", dam = "dam", sire = "sire", keep_names = FALSE)
```
Arguments

ped  A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.

id  A string naming the column with individual identities. It will be renamed to its default value 'id'.

dam  A string naming the column with maternal identities. It will be renamed to its default value 'dam'.

sire  A string naming the column with paternal identities. It will be renamed to its default value 'sire'.

keep_names  A boolean value indicating whether the original identity values should be kept on a separate column (named 'names'), or not.

Value

A sorted pedigree dataframe (with ancestors on top of descendants).

References


See Also

ped_rename

Examples

data(darwin)
# Here we reshuffle rows in the pedigree. It won't be usable for other functions in the package
darwin <- darwin[sample(1:nrow(darwin)), ]
# Below, we sort the pedigree again. The order might not be the same as before.
# But ancestors will always be placed on top of descendants,
# making the pedigree usable for other functions in the package.
darwin <- ped_sort(darwin, id = "Individual", dam = "Mother", sire = "Father", keep_names = TRUE)

Recursive function that computes steps for sorting algorithm described by Zhang et al (2009).

Usage

sort_step(p, id, dam, sire, t, S, G, t_G)
Arguments

- **p**: Pedigree to sort (used as template)
- **id**: A string naming the column with individual identities. It will be renamed to its default value 'id'.
- **dam**: A string naming the column with maternal identities. It will be renamed to its default value 'dam'.
- **sire**: A string naming the column with paternal identities. It will be renamed to its default value 'sire'.
- **t**: Template for the new sorted pedigree
- **S**: Vector of assumed parent individuals
- **G**: Vector of generation numbers (0 identifies the youngest)
- **t_G**: Vector G for the new sorted pedigree

Value

No return value. Will print an error message if checking fail.
Filled template for the sorted pedigree. Once recursion ends, it returns the sorted pedigree

References


See Also

- `ped_sort`

---

**pop_hwd**

*Deviations from Hardy-Weinberg equilibrium*

Description

Computes the deviation from Hardy-Weinberg equilibrium following Caballero and Toro (2000).

Usage

`pop_hwd(ped, reference = NULL)`

Arguments

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **reference**: A string naming a column indicating whether individuals belong to the reference population or not. Column must be boolean or coercible to boolean type.
Value

A numeric value indicating the deviation from Hardy-Weinberg equilibrium.

References


See Also

pop_Ne

Examples

```r
data(atlas)
pop_hwd(dama)
```

Description

Estimate the total and effective number of founders and ancestors in a pedigree, as well as the number of founder genome equivalents (see details on these parameters below). Note that a reference population (RP) must be defined, so that founders and ancestors are referred to the set of individuals belonging to that RP. This is set by means of a boolean vector passed as argument.

Usage

```r
pop_Nancestors(ped, reference, nboot = 10000L, seed = NULL, skip_Ng = FALSE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ped</td>
<td>A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.</td>
</tr>
<tr>
<td>reference</td>
<td>A string naming a column indicating whether individuals belong to the reference population or not. Column must be boolean or coercible to boolean type.</td>
</tr>
<tr>
<td>nboot</td>
<td>Number of bootstrap iterations (for computing Ng).</td>
</tr>
<tr>
<td>seed</td>
<td>Sets a seed for the random number generator.</td>
</tr>
<tr>
<td>skip_Ng</td>
<td>Skip Ng computation or not (FALSE by default).</td>
</tr>
</tbody>
</table>
Details

The total number of founders \((N_f)\) and ancestors \((N_a)\) are calculated simply as the count of founders and ancestors of individuals belonging to the reference population (RP). Founders here are defined as individuals with both parentals unknown.

The effective number of founders \((N_{fe})\) is the number of equally contributing founders, that would account for observed genetic diversity in the RP, while the effective number of ancestors \((N_{ae})\) is defined as the minimum number of ancestors, founders or not, required to account for the genetic diversity observed in the RP. These parameters are computed from the probability of gene origin, following methods in Tahmoorespur and Sheikhloo (2011).

While the previous parameters account for diversity loss due to bottlenecks at the level of founders or ancestors, other sources of random loss of alleles (such as drift) can be accounted by means of the number of founder genome equivalents \((N_g)\), Caballero and Toro (2000). This parameter is estimated via Monte Carlo simulation of allele segregation, following Boichard et al. (1997).

Value

A dataframe containing population size estimates for founders and ancestors:

- \(N_r\) - Total number of individuals in the RP
- \(N_f\) - Total number of founders
- \(N_{fe}\) - Effective number of founders
- \(N_a\) - Total number of ancestors
- \(N_{ae}\) - Effective number of ancestors
- \(N_g\) - Number of founder genome equivalents
- \(se_{N_g}\) - Standard error of Ng

If some of the auxiliary functions is used (e.g. \(pop_Nr\)), only the corresponding numerical estimate will be returned. In the case of \(pop_Ng\), a list object is returned, with the number of founder genome equivalents \((N_g)\) and its standard error \((se_{N_g})\).

References


Examples

data(arrui)
pop_Nancestors(arrui, reference = "target", skip_Ng = TRUE)
Effective population size

Description

Estimate the effective population size \((N_e)\). This is computed from the increase in individual inbreeding, following the method described by Gutiérrez et al (2008, 2009).

Usage

\[
\text{pop\_Ne}(\text{ped}, \text{Fcol}, \text{tcol})
\]

Arguments

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **Fcol**: Name of column with inbreeding coefficient values.
- **tcol**: Name of column with generation numbers.

Value

A list with the effective population size \((N_e)\) and its standard error \((\text{se}\_N_e)\).

References


See Also

- `ip\_F`, `pop\_t`

Examples

```
data(atlas)
atlas <- ip\_F(atlas) # compute inbreeding, appending column "F"
atlas <- pop\_t(atlas) # compute generations, appending column "t"
pop\_Ne(atlas, Fcol = "Fi", tcol = "t")
```
Description

Computes the number of equivalent complete generations \((t)\), as defined by Boichard et al (1997).

Usage

\[
pop_t(ped, \text{name_to} = "t")
\]

Arguments

- **ped** A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **name_to** A string naming the new output column.

Value

The input dataframe, plus an additional column corresponding to the number of equivalent complete generations of every individual (named "t" by default).

References


See Also

- **pop_Ne**

Examples

```r
data(dama)
dama <- pop_t(dama)
tail(dama)
```
purgeR: Estimation of inbreeding-purging genetic parameters in pedigreed populations

Description

The purgeR package includes functions for the computation of parameters related to inbreeding and genetic purging in pedigreed populations, including standard, ancestral and purged inbreeding coefficients, among other measures of inbreeding and purging. In addition, functions to compute the effective population size and other parameters relevant to population genetics and structure are included.

Details

A complete user’s guide with examples is provided as vignettes, introducing functions in this package and providing examples of use. Navigate these vignettes from R with:

```
browseVignettes("purgeR")
```

There are currently two vignettes available:

- **purgeR-tutorial**: A complete overview of all functions in the package, including easy to follow examples.
- **ip**: A more advanced guide showing examples of inbreeding purging analyses.

Functions

- **Preprocessing**
  - `ped_rename` - Rename individuals in a pedigree from 1 to N
  - `ped_sort` - Sort individuals (with ancestors on top of descendants)
  - `ped_clean` - Remove individuals not used for purging analyses
  - `ped_maternal` - Maternal effects

- **Inbreeding and purging**
  - `ip_F` - Inbreeding coefficient
  - `ip_Fa` - Ancestral inbreeding coefficient
  - `ip_Fij` - Partial inbreeding coefficient
  - `ip_g` - Purged inbreeding coefficient
  - `ip_op` - Opportunity of purging
  - `exp_F` - Expected inbreeding coefficient
  - `exp_Fa` - Expected ancestral inbreeding coefficient
  - `exp_g` - Expected purged inbreeding coefficient

- **Population parameters**
  - `pop_hwd` - Deviation from Hardy-Weinberg equilibrium
  - `pop_t` - Number of equivalent complete generations
  - `pop_Ne` - Effective population size
  - `pop_Nancestors` - Population founders and ancestors
  - `pop_Na` - Total number of ancestors
  - `pop_Nae` - Effective number of ancestors
  - `pop_Nf` - Total number of founders
  - `pop_Nfe` - Effective number of founders
  - `pop_Ng` - Number of founder genome equivalents

- **Fitness**
  - `w_grandoffspring` - Grandoffspring
  - `w_offspring` - Offspring
  - `w_reproductive_value` - Reproductive value

Author(s)

Eugenio López-Cortegano <elcortegano@gmail.com> (ORCID)
References


See Also

Source code is available via the GitLab repository at https://gitlab.com/elcortegano/purgeR/. Users are encouraged to report bugs, request features, and contribute code to this project. Some users might find useful the C++ software PURGd, which computes inbreeding-purging parameters and follow-up statistical analyses: https://gitlab.com/elcortegano/PURGd/.

---

reproductive_value

Reproductive value

Description

Computes the reproductive value

Usage

```R
reproductive_value(
  ped,
  reference,
  name_to,
  target = NULL,
  enable_correction = TRUE
)
```

Arguments

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **reference**: A string naming a column indicating whether individuals belong to the reference population or not. Column must be boolean or coercible to boolean type.
- **name_to**: A string naming the new output column.
- **target**: A string naming a column indicating whether individuals belong to the target population or not. Column must be boolean or coercible to boolean type. By default, all descendants of the reference population are used.
- **enable_correction**: Correct reproductive values.

Value

The input dataframe, plus an additional column with reproductive values for the reference and target populations assumed.
References


---

**sample_allele**

*Sample dam or sire inherited allele*

**Description**

Given two alleles (one from dam, the other from sire), it samples one at random.

**Arguments**

- dam_al: Dam allele.
- sire_al: Sire allele.

**Value**

The sampled allele.

---

**search_ancestors**

*Search and individuals’ ancestors*

**Description**

Recursive function that gathers all founders and ancestors for a given individual

**Arguments**

- dam: Vector of dams.
- sire: Vector of sires.
- i: Reference individual (its index, not id).
- fnd: Vector of founders (to be returned as reference).
- anc: Vector of ancestors (to be returned as reference).

**Value**

The sampled allele.
### w_grandoffspring

**Grandoffspring**

**Description**
Counts the number of grandoffspring for individuals in the pedigree.

**Usage**

```
w_grandoffspring(ped, name_to)
```

**Arguments**

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **name_to**: A string naming the new output column.

**Value**

The input dataframe, plus an additional column indicating the total number of grandoffspring.

**Examples**

```
data(arrui)
dama <- w_grandoffspring(arrui, name_to = "G")
head(arrui)
```

### w_offspring

**Offspring**

**Description**
Counts the number of offspring for individuals in the pedigree.

**Usage**

```
w_offspring(ped, name_to, dam_offspring = TRUE, sire_offspring = TRUE)
```

**Arguments**

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **name_to**: A string naming the new output column.
- **dam_offspring**: Compute dam offspring (TRUE by default).
- **sire_offspring**: Compute sire offspring (TRUE by default).
**w_reproductive_value**

**Value**

The input dataframe, plus an additional column indicating the total number of offspring.

**Examples**

```r
data(arrui)
dama <- w_offspring(arrui, name_to = "P")
head(arrui)
```

---

**Description**

Computes the reproductive value following the method by Hunter et al. (2019). This is a measure of how well a gene originated in a set of ‘reference’ individuals is represented in a different set of ‘target’ individuals. By default, fitness is computed for individuals in the reference population, using all of their descendants as target. A generation-wise mode can also be enabled, to compute fitness contributions consecutively from one generation to the next.

**Usage**

```r
w_reproductive_value(
  ped,
  reference,
  name_to,
  target = NULL,
  enable_correction = TRUE,
  generation_wise = FALSE
)
```

**Arguments**

- **ped**: A dataframe containing the pedigree. Individual (id), maternal (dam), and paternal (sire) identities are mandatory columns.
- **reference**: A string naming a column indicating whether individuals belong to the reference population or not. Column must be boolean or coercible to boolean type.
- **name_to**: A string naming the new output column.
- **target**: A string naming a column indicating whether individuals belong to the target population or not. Column must be boolean or coercible to boolean type. By default, all descendants of the reference population are used.
- **enable_correction**: Correct reproductive values (enabled by default).
- **generation_wise**: Assume that the reference population is a vector of integers indicating generation numbers. Reproductive values will be computed generation by generation independently (except for the last one).
Details

A reference population must be defined, which represents a set of individuals whose reproductive value is to be calculated. By default, genetic contributions to the rest of individuals in the pedigree is assumed, but a target population can also be defined, restricting the set of individuals accounted when computing the reproductive value. This could represent for example a cohort of alive individuals.

Value

The input dataframe, plus an additional column with reproductive values for the reference and target populations assumed.

References


Examples

```r
library(dplyr)
library(magrittr)

# Pedigree used in Hunter et al. (2019)
       "B1", "B2", "B3", "B4",
       "C1", "C2", "C3", "C4")
dam <- c("O", "O", "O", "O", "O", "O",
sire <- c("O", "O", "O", "O", "O", "O",
t <- c(0, 0, 0, 0, 0, 0,
      1, 1, 1, 1,
      2, 2, 2, 2)
ped <- tibble::tibble(id, dam, sire, t)
ped <- purgeR::ped_rename(ped, keep_names = TRUE) %>%
  dplyr::mutate(reference = ifelse(t == 1, TRUE, FALSE))
purgeR::w_reproductive_value(ped, reference = "reference", name_to = "R")
```
Index

* datasets
  arrui, 4
  atlas, 5
  dama, 17
  darwin, 18
  dorcas, 19

ancestors, 3
arrui, 4
atlas, 5

check_ancestors, 6
check_basic, 6
check_bool, 7
check_col, 8
check_d, 8
check_df, 9
check_Fcol, 9
check_index, 10
check_int, 10
check_length, 11
check_na, 11
check_names, 12
check_F, 12
check_not_col, 13
check_nrows, 13
check_order, 14
check_reference, 14
check_repeat_id, 15
check_target (check_reference), 14
check_tcol, 15
check_types, 16
check_zero_id, 16

dama, 17
darwin, 18
delta_Fi, 18
dorcas, 19

exp_F, 20, 28, 45
exp_Fa, 21, 29, 45
exp_g, 22, 31, 45

F, 23
Fa, 23
Fij_core, 24
Fij_core_i_cpp, 25
g, 25

hwd, 26

idx_ancestors, 27
ip_F, 20, 27, 29–31, 43, 45
ip_Fa, 21, 28, 45
ip_Fij, 29, 33, 45
ip_g, 22, 31, 45
ip_op, 32, 45

map_ancestors, 34

Ne_delta, 34

op, 35

ped_clean, 36, 38, 45
ped_maternal, 37, 45
ped_rename, 36, 37, 38, 39, 45
ped_sort, 38, 40, 45
ped_sort_i, 39
pop_hwd, 40, 45
pop_Na, 45
pop_Na (pop_Nancestors), 41
pop_Nae, 45
pop_Nae (pop_Nancestors), 41
pop_Nancestors, 41, 45
pop_Ne, 27, 41, 43, 44, 45
pop_Nf, 45
pop_Nf (pop_Nancestors), 41
pop_Nfe, 45
pop_Nfe (pop_Nancestors), 41
pop_Ng, 45
pop_Ng (pop_Nancestors), 41
pop_t, 43, 44, 45
purgeR, 45
purgeR-package (purgeR), 45
reproductive_value, 46
sample_allele, 47
se_Ne_delta (Ne_delta), 34
search_ancestors, 47
sort_step (ped_sort_i), 39
w_grandoffspring, 45, 48
w_offspring, 45, 48
w_reproductive_value, 45, 49