Package: Pedixplorer (via r-universe)

June 3, 2026

Version 1.8.0

Date 2025-08-26

Title Pedigree Functions

Depends R (>= 4.4.0)

Imports graphics, stats, methods, ggplot2, utils, grDevices, stringr, plyr, dplyr, tidyr, quadprog, Matrix, S4Vectors, shiny, readxl, DT, igraph, shinycssloaders, shinyhelper, shinyjs, shinyjqui, shinyWidgets, htmlwidgets, plotly, colourpicker, shinytoastr

Description Routines to handle family data with a Pedigree object. The initial purpose was to create correlation structures that describe family relationships such as kinship and identity-by-descent, which can be used to model family data in mixed effects models, such as in the coxme function. Also includes a tool for Pedigree drawing which is focused on producing compact layouts without intervention. Recent additions include utilities to trim the Pedigree object with various criteria, and kinship for the X chromosome.

License Artistic-2.0

Encoding UTF-8

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Suggests diffviewer, gridExtra, testthat (>= 3.0.0), vdiffr, rmarkdown, BiocStyle, knitr, withr, qpdf, shinytest2, devtools, R.devices, usethis, rlang, magick, cowplot

Config/testthat/edition 3

biocViews Software, DataRepresentation, Genetics, GraphAndNetwork, Visualization

BugReports https://github.com/LouisLeNezet/Pedixplorer/issues

URL https://louislenezet.github.io/Pedixplorer/

BiocType Software

Collate 'AllValidity.R' 'AllClass.R' 'kindepth.R' 'kinship.R'
'utils.R' 'AllConstructor.R' 'AllAccessors.R' 'AllGeneric.R'
'Pedixplorer-package.R' 'alignped4.R' 'alignped3.R'
'alignped2.R' 'alignped1.R' 'auto_hint.R' 'align.R' 'app.R'
'app_color_picker.R' 'app_data_col_sel.R' 'app_data_download.R'
'app_data_import.R' 'app_utils.R' 'app_family_sel.R'
'app_health_sel.R' 'app_inf_sel.R' 'app_ped_avaf_infos.R'
'app_plot_resize.R' 'app_plot_download.R' 'app_plot_legend.R'
'app_plot_ped.R' 'app_plot_all.R' 'app_server.R' 'app_ui.R'
'best_hint.R' 'bit_size.R' 'data.R' 'descendants.R'
'family_check.R' 'find_unavailable.R' 'find_avail_affected.R'
'find_avail_noninform.R' 'fix_parents.R' 'generate_aff_inds.R'
'generate_colors.R' 'ibd_matrix.R' 'is_informative.R'
'make_famid.R' 'min_dist_inf.R' 'norm_data.R' 'num_child.R'
'ped_to_legdf.R' 'ped_to_plotdf.R' 'plot_fct.R' 'plot_fromdf.R'
'plot_pedigree.R' 'shrink.R' 'unrelated.R' 'useful_inds.R'

LazyData false

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libuv1-dev libxml2-dev libssl-dev zlib1g-dev

Repository https://bioc-release.r-universe.dev

Date/Publication 2026-04-28 13:02:23 UTC

RemoteUrl https://github.com/bioc/Pedixplorer

RemoteRef RELEASE_3_23

RemoteSha 98d7b527b4650bfbb568ed1121f73059784a6356

Contents

Pedixplorer-package . . . 3
align . . . 5
alignped1 . . . 7
alignped2 . . . 9
alignped3 . . . 10
alignped4 . . . 12
auto_hint . . . 13
best_hint . . . 15
bit_size . . . 17
family_infos_table . . . 18
find_avail_affected . . . 18
find_avail_noninform . . . 19
find_unavailable . . . 20
fix_parents . . . 22
generate_colors . . . 23
Hints-class . . . 26
ibd_matrix . . . 28
is_informative . . . 29

is_parent . . . 31
kindepth . . . 32
kinship . . . 33
make_famid . . . 35
min_dist_inf . . . 36
minnbreast . . . 38
norm_ped . . . 40
norm_rel . . . 42
num_child . . . 44
parent_of . . . 45
Ped-class . . . 46
ped_server . . . 52
ped_shiny . . . 53
ped_to_legdf . . . 54
ped_to_plotdf . . . 55
ped_ui . . . 58
Pedigree-class . . . 58
plink_to_pedigree . . . 65
plot,Pedigree,missing-method . . . 66
plot_fromdf . . . 70
Rel-class . . . 71
relped . . . 74
sampleped . . . 75
Scales-class . . . 76
shrink . . . 78
unrelated . . . 80
upd_famid . . . 81
useful_inds . . . 82

Index **84**

`Pedixplorer-package` *The Pedixplorer package for pedigree data*

Description

The Pedixplorer package for pedigree data an updated package of the `kinship2` package. The `kinship2` package was originally written by Terry Therneau and Jason Sinnwell. The `Pedixplorer` package is a fork of the `kinship2` package with additional functionality and bug fixes.

Details

The package download, NEWS, and README are available on CRAN: Kinship2 for the previous version of the package.

Functions

Below are listed some of the most widely used functions available in arsenal:

`Pedigree()`: Contstructor of the Pedigree class, given identifiers, sex, affection status(es), and special relationships

`kinship()`: Calculates the kinship matrix, the probability having an allele sampled from two individuals be the same via IBD.

`plot()` : Method to transform a Pedigree object into a graphical plot. Allows extra information to be included in the id under the plot symbol. This method use the `plot_fromdf()` function to transform the Pedigree object into a data frame of graphical elements, the same is done for the legend with the `ped_to_legdf()` function. When done, the data frames are plotted with the `plot_fromdf()` function.

`shrink()`: Shrink a Pedigree to a specific bit size, removing non-informative members first.

`bit_size()`: Approximate the output from SAS's `PROC FREQ` procedure when using the `/list` option of the `TABLE` statement.

Data

- `sampleped()`: Pedigree example data sets with two pedigrees
- `minnbreast()`: Larger cohort of pedigrees from MN breast cancer study

Author(s)

Maintainer: Louis Le Nezet <louislenezet@gmail.com> (ORCID) [contributor]

Authors:

- Jason Sinnwell <sinnwell.jason@mayo.edu>
- Terry Therneau

Other contributors:

- Daniel Schaid [contributor]
- Elizabeth Atkinson [contributor]

See Also

Useful links:

- https://louislenezet.github.io/Pedixplorer/
- Report bugs at https://github.com/LouisLeNezet/Pedixplorer/issues

Examples

```
library(Pedixplorer)
```

`align` *Align a Pedigree object*

Description

Given a Pedigree, this function creates helper matrices that describe the layout of a plot of the Pedigree.

Usage

```
## S4 method for signature 'Pedigree'
align(
  obj,
  packed = TRUE,
  width = 10,
  align = TRUE,
  hints = NULL,
  missid = "NA_character_",
  align_parents = TRUE,
  force = FALSE,
  precision = 4
)
```

Arguments

`obj`	A Pedigree object
`packed`	Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
`width`	For a packed output, the minimum width of the plot, in inches.
`align`	For a packed Pedigree, align children under parents `TRUE`, to the extent possible given the page width, or align to to the left margin `FALSE`. This argument can be a two element vector, giving the alignment parameters, or a logical value. If `TRUE`, the default is `c(1.5, 2)`, or if numeric the routine `alignped4()` will be called.
`hints`	A Hints object or a named list containing `horder` and `spouse`. If `NULL` then the Hints stored in **obj** will be used.
`missid`	A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to `NA_character_`.
`align_parents`	If `align_parents = TRUE`, go one step further and try to make both parents of each child have the same depth. (This is not always possible). It helps the drawing program by lining up pedigrees that 'join in the middle' via a marriage.
`force`	If `force = TRUE`, the function will return the depth minus `min(depth)` if `depth` reach a state with no founders is not possible.
`precision`	The number of significatif numbers to round the solution to.

Details

This is an internal routine, used almost exclusively by `ped_to_plotdf()`.

The subservient functions `auto_hint()`, `alignped1()`, `alignped2()`, `alignped3()`, and `alignped4()` contain the bulk of the computation.

If the **hints** are missing the `auto_hint()` routine is called to supply an initial guess.

If multiple families are present in the **obj** Pedigree, this routine is called once for each family, and the results are combined in the list returned.

For more information you can read the associated vignette: `vignette("pedigree_alignment")`.

Value

A list with components

- `n`: A vector giving the number of subjects on each horizonal level of the plot
- `nid`: A matrix with one row for each level, giving the numeric id of each subject plotted. (A value of 17 means the 17th subject in the Pedigree).
- `pos`: A matrix giving the horizontal position of each plot point
- `fam`: A matrix giving the family id of each plot point. A value of 3 would mean that the two subjects in positions 3 and 4, in the row above, are this subject's parents.
- `spouse`: A matrix with values
  - `0` = not a spouse
  - `1` = subject plotted to the immediate right is a spouse
  - `2` = subject plotted to the immediate right is an inbred spouse
- `twins`: Optional matrix which will only be present if the Pedigree contains twins :
  - `0` = not a twin
  - `1` = sibling to the right is a monozygotic twin
  - `2` = sibling to the right is a dizygotic twin
  - `3` = sibling to the right is a twin of unknown zygosity

See Also

`alignped1()`, `alignped2()`, `alignped3()`, `alignped4()`, `auto_hint()`

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
align(pedi)
```

`alignped1` *Alignment first routine*

Description

First alignment routine which create the subtree founded on a single subject as though it were the only tree.

Usage

```
alignped1(idx, dadx, momx, level, horder, packed, spouselist)
```

Arguments

`idx`	Indexes of the subjects
`dadx`	Indexes of the fathers
`momx`	Indexes of the mothers
`level`	Vector of the level of each subject
`horder`	A named numeric vector with one element per subject in the Pedigree. It determines the relative horizontal order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters). The names of the vector should be the individual identifiers.
`packed`	Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
`spouselist`	Matrix of spouses with 4 columns: • `1`: husband index • `2`: wife index • `3`: husband anchor • `4`: wife anchor

Details

In this routine the **nid** array consists of the final `nid array + 1/2` of the final spouse array. Note that the **spouselist** matrix will only contain spouse pairs that are not yet processed. The logic for anchoring is slightly tricky.

1. Anchoring::

First, if col 4 of the spouselist matrix is 0, we anchor at the first opportunity. Also note that if `spouselist[, 3] == spouselist[, 4]` it is the husband who is the anchor (just write out the possibilities).

2. Return values initialization::

Create the set of 3 return structures, which will be matrices with `1 + nspouse` columns. If there are children then other routines will widen the result.

3. Create lspouse and rspouse::

This two complimentary lists denote the spouses plotted on the left and on the right. For someone with lots of spouses we try to split them evenly. If the number of spouses is odd, then men should have more on the right than on the left, women more on the right. Any hints in the spouselist matrix override. We put the undecided marriages closest to **idx**, then add predetermined ones to the left and right. The majority of marriages will be undetermined singletons, for which **nleft** will be `1` for female (put my husband to the left) and `0` for male. In one bug found by plotting canine data, lspouse could initially be empty but `length(rspouse) > 1`. This caused `nleft > length(indx)`. A fix was to not let **indx** to be indexed beyond its length, fix by JPS 5/2013.

4. List the children::

For each spouse get the list of children. If there are any we call `alignped2()` to generate their tree and then mark the connection to their parent. If multiple marriages have children we need to join the trees.

5. Splice the tree::

To finish up we need to splice together the tree made up from all the kids, which only has data from `lev + 1` down, with the data here. There are 3 cases:

1. No children were found.
2. The tree below is wider than the tree here, in which case we add the data from this level onto theirs.
3. The tree below is narrower, for instance an only child.

Value

A list containing the elements to plot the Pedigree. It contains a set of matrices along with the spouselist matrix. The latter has marriages removed as they are processed.

- `n` : A vector giving the number of subjects on each horizonal level of the plot
- `nid` : A matrix with one row for each level, giving the numeric id of each subject plotted. (A value of `17` means the 17th subject in the Pedigree).
- `pos` : A matrix giving the horizontal position of each plot point
- `fam` : A matrix giving the family id of each plot point. A value of `3` would mean that the two subjects in positions 3 and 4, in the row above, are this subject's parents.
- `spouselist` : Spouse matrix with anchors informations

See Also

`align()`

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
align(pedi)
```

`alignped2` *Alignment second routine*

Description

Second of the four co-routines which takes a collection of siblings, grows the tree for each, and appends them side by side into a single tree.

Usage

```
alignped2(idx, dadx, momx, level, horder, packed, spouselist)
```

Arguments

`idx`	Indexes of the subjects
`dadx`	Indexes of the fathers
`momx`	Indexes of the mothers
`level`	Vector of the level of each subject
`horder`	A named numeric vector with one element per subject in the Pedigree. It determines the relative horizontal order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters). The names of the vector should be the individual identifiers.
`packed`	Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
`spouselist`	Matrix of spouses with 4 columns: • 1: husband index • 2: wife index • 3: husband anchor • 4: wife anchor

Details

The input arguments are the same as those to `alignped1()` with the exception that **idx** will be a vector. This routine does nothing to the spouselist matrix, but needs to pass it down the tree and back since one of the routines called by `alignped2()` might change the matrix.

The code below has one non-obvious special case. Suppose that two sibs marry. When the first sib is processed by `alignped1` then both partners (and any children) will be added to the rval structure below. When the second sib is processed they will come back as a 1 element tree (the marriage will no longer be on the **spouselist**), which should be added onto rval. The rule thus is to not add any 1 element tree whose value (which must be `idx[i]` is already in the rval structure for this level.

Value

A list containing the elements to plot the Pedigree. It contains a set of matrices along with the spouselist matrix. The latter has marriages removed as they are processed.

- `n` : A vector giving the number of subjects on each horizonal level of the plot
- `nid` : A matrix with one row for each level, giving the numeric id of each subject plotted. (A value of 17 means the 17th subject in the Pedigree).
- `pos` : A matrix giving the horizontal position of each plot point
- `fam` : A matrix giving the family id of each plot point. A value of 3 would mean that the two subjects in positions 3 and 4, in the row above, are this subject's parents.
- `spouselist` : Spouse matrix with anchors informations

See Also

`align()`

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
align(pedi)
```

`alignped3` *Alignment third routine*

Description

Third of the four co-routines to merges two pedigree trees which are side by side into a single object.

Usage

```
alignped3(alt1, alt2, packed, space = 1)
```

Arguments

`alt1`	Alignment of the first tree
`alt2`	Alignment of the second tree
`packed`	Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
`space`	Space between two subjects

Details

The primary special case is when the rightmost person in the left tree is the same as the leftmost person in the right tree; we need not plot two copies of the same person side by side. (When initializing the output structures do not worry about this, there is no harm if they are a column bigger than finally needed.) Beyond that the work is simple book keeping.

1. Slide::

For the unpacked case, which is the traditional way to draw a Pedigree when we can assume the paper is infinitely wide, all parents are centered over their children. In this case we think if the two trees to be merged as solid blocks. On input they both have a left margin of 0. Compute how far over we have to slide the right tree.

2. Merge::

Now merge the two trees. Start at the top level and work down.

Value

A list containing the elements to plot the Pedigree. It contains a set of matrices along with the spouselist matrix. The latter has marriages removed as they are processed.

- `n` : A vector giving the number of subjects on each horizonal level of the plot
- `nid` : A matrix with one row for each level, giving the numeric id of each subject plotted. (A value of 17 means the 17th subject in the Pedigree).
- `pos` : A matrix giving the horizontal position of each plot point
- `fam` : A matrix giving the family id of each plot point. A value of 3 would mean that the two subjects in positions 3 and 4, in the row above, are this subject's parents.
- `spouselist` : Spouse matrix with anchors informations

See Also

`align()`

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
align(pedi)
```

alignped4 *Alignment fourth routine*

Description

Last routines which attempts to line up children under parents and put spouses and siblings "close" to each other, to the extent possible within the constraints of page width.

Usage

```
alignped4(rval, spouse, level, width, align, precision = 4)
```

Arguments

rval	A list with components n, nid, pos, and fam.
spouse	A boolean matrix with one row per level representing if the subject is a spouse or not.
level	Vector of the level of each subject
width	For a packed output, the minimum width of the plot, in inches.
align	For a packed Pedigree, align children under parents TRUE, to the extent possible given the page width, or align to to the left margin FALSE. This argument can be a two element vector, giving the alignment parameters, or a logical value. If TRUE, the default is c(1.5, 2), or if numeric the routine alignped4() will be called.
precision	The number of significatif numbers to round the solution to.

Details

The alignped4() routine is the final step of alignment. The current code does necessary setup and then calls the quadprog::solve.QP() function.

There are two important parameters for the function:

1. The maximum width specified. The smallest possible width is the maximum number of subjects on a line. If the user suggestion is too low it is increased to that amount plus one (to give just a little wiggle room).
2. The align vector of 2 alignment parameters a and b. For each set of siblings x with parents at p_1 and p_2 the alignment penalty is:

$$(1/k^a)\sum_{i=1}^{k}(x_i - (p_1 + p_2)/2)^2$$

where k is the number of siblings in the set.

Using the fact that when a = 1 :

$$\sum(x_i - c)^2 = \sum(x_i - \mu)^2 + k(c - \mu)^2$$

then moving a sibship with k sibs one unit to the left or right of optimal will incur the same cost as moving one with only 1 or two sibs out of place.

If a = 0 then large sibships are harder to move than small ones. With the default value a = 1.5, they are slightly easier to move than small ones. The rationale for the default is as long as the parents are somewhere between the first and last siblings the result looks fairly good, so we are more flexible with the spacing of a large family. By tethering all the sibs to a single spot they tend to be kept close to each other.

The alignment penalty for spouses is $b(x_1 - x_2)^2$, which tends to keep them together. The size of b controls the relative importance of sib-parent and spouse-spouse closeness.

1. We start by adding in these penalties. The total number of parameters in the alignment problem (what we hand to quadprog) is the set of sum(n) positions. A work array myid keeps track of the parameter number for each position so that it is easy to find. There is one extra penalty added at the end. Because the penalty amount would be the same if all the final positions were shifted by a constant, the penalty matrix will not be positive definite; solve.QP() does not like this. We add a tiny amount of leftward pull to the widest line.
2. If there are k subjects on a line there will be k+1 constraints for that line. The first point must be ≥ 0, each subsequent one must be at least 1 unit to the right, and the final point must be $\leq$ the max width.

Value

The updated position matrix

See Also

align()

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
align(pedi)
```

---

auto_hint *Initial hint for a Pedigree alignment*

Description

Compute an initial guess for the alignment of a Pedigree

Usage

```
## S4 method for signature 'Pedigree'
auto_hint(
  obj,
  hints = NULL,
  packed = TRUE,
  align = FALSE,
  reset = FALSE,
  align_parents = TRUE,
  force = FALSE
)
```

Arguments

obj	A Pedigree object
hints	A Hints object or a named list containing horder and spouse. If NULL then the Hints stored in **obj** will be used.
packed	Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
align	For a packed Pedigree, align children under parents TRUE, to the extent possible given the page width, or align to to the left margin FALSE. This argument can be a two element vector, giving the alignment parameters, or a logical value. If TRUE, the default is c(1.5, 2), or if numeric the routine alignped4() will be called.
reset	If TRUE, then even if the Ped object has Hints, reset them to the initial values.
align_parents	If align_parents = TRUE, go one step further and try to make both parents of each child have the same depth. (This is not always possible). It helps the drawing program by lining up pedigrees that 'join in the middle' via a marriage.
force	If force = TRUE, the function will return the depth minus min(depth) if depth reach a state with no founders is not possible.

Details

A Pedigree structure can contain a Hints object which helps to reorder the Pedigree (e.g. left-to-right order of children within family) so as to plot with minimal distortion. This routine is used to create an initial version of the hints. They can then be modified if desired.

This routine would not normally be called by a user. It moves children within families, so that marriages are on the "edge" of a set children, closest to the spouse. For pedigrees that have only a single connection between two families this simple-minded approach works surprisingly well. For more complex structures hand-tuning of the hints may be required.

When auto_hint() is called with a a vector of numbers as the **hints** argument, the values for the founder females are used to order the founder families left to right across the plot. The values within a sibship are used as the preliminary order of siblings within a family; this may be changed to move one of them to the edge so as to match up with a spouse. The actual values in the vector are not important, only their order.

Value

The initial Hints object.

See Also

`align()`, `best_hint()`

Hints

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped[sampleped$famid == 1, ])
Pedixplorer:::auto_hint(pedi)
```

---

best_hint *Best hint for a Pedigree alignment*

Description

When computer time is cheap, use this routine to get a *best* Pedigree alignment. This routine will try all possible founder orders, and return the one with the least **stress**.

Usage

```
## S4 method for signature 'Pedigree'
best_hint(
  obj,
  wt = c(1000, 10, 1),
  tolerance = 0,
  align_parents = TRUE,
  force = FALSE,
  timeout = 60
)
```

Arguments

`obj`	A Pedigree object
`wt`	A vector of three weights for the three error measures. Default is `c(1000, 10, 1)`. 1. The number of duplicate individuals in the plot 2. The sum of the absolute values of the differences in the positions of duplicate individuals 3. The sum of the absolute values of the differences between the center of the children and the parents.
`tolerance`	The maximum stress level to accept. Default is `0`

`align_parents`	If `align_parents = TRUE`, go one step further and try to make both parents of each child have the same depth. (This is not always possible). It helps the drawing program by lining up pedigrees that 'join in the middle' via a marriage.
`force`	If `force = TRUE`, the function will return the depth minus `min(depth)` if `depth` reach a state with no founders is not possible.
`timeout`	The maximum time in seconds to spend searching for the best hint. Default is `60` seconds.

Details

The `auto_hint()` routine will rearrange sibling order, but not founder order. This calls `auto_hint()` with every possible founder order, and finds that plot with the least "stress". The stress is computed as a weighted sum of three error measures:

- nbArcs The number of duplicate individuals in the plot
- lgArcs The sum of the absolute values of the differences in the positions of duplicate individuals
- lgParentsChilds The sum of the absolute values of the differences between the center of the children and the parents

$$stress = wt[1] * nbArcs + wt[2] * lgArcs + wt[3] * lgParentsChilds$$

If during the search, a plot is found with a stress level less than **tolerance**, the search is terminated.

Value

The best Hints object out of all the permutations

See Also

`auto_hint()`, `align()`

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped[sampleped$famid == 1,])
best_hint(pedi)
```

bit_size *Bit size of a Pedigree*

Description

Utility function used in the shrink() function to calculate the bit size of a Pedigree.

Usage

```
## S4 method for signature 'character_OR_integer'
bit_size(obj, momid, missid = NA_character_)

## S4 method for signature 'Pedigree'
bit_size(obj)

## S4 method for signature 'Ped'
bit_size(obj)
```

Arguments

obj	A Ped or Pedigree object or a vector of fathers identifiers
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
missid	A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

Details

The bit size of a Pedigree is defined as :

$$2 \times NbNonFounders - NbFounders$$

Where NbNonFounders is the number of non founders in the Pedigree (i.e. individuals with identified parents) and NbFounders is the number of founders in the Pedigree (i.e. individuals without identified parents).

Value

A list with the following components:

- bit_size The bit size of the Pedigree
- nFounder The number of founders in the Pedigree
- nNonFounder The number of non founders in the Pedigree

See Also

shrink()

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
bit_size(pedi)
```

`family_infos_table` *Affection and availability information table*

Description

This function creates a table with the affection and availability information for all individuals in a pedigree object.

Usage

```
family_infos_table(pedi, col_val = NA)
```

Arguments

`pedi`	A pedigree object.
`col_val`	The column name in the `fill` slot of the pedigree object to use for the table.

Value

A cross table dataframe with the affection and availability information.

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
pedi <- generate_colors(pedi, "num_child_tot", threshold = 2)
Pedixplorer:::family_infos_table(pedi, "num_child_tot")
Pedixplorer:::family_infos_table(pedi, "affection")
```

`find_avail_affected` *Find single affected and available individual from a Pedigree*

Description

Finds one subject from among available non-parents with indicated affection status.

Usage

```
## S4 method for signature 'Ped'
find_avail_affected(obj, avail = NULL, affected = NULL, affstatus = NA)

## S4 method for signature 'Pedigree'
find_avail_affected(obj, avail = NULL, affected = NULL, affstatus = NA)
```

Arguments

obj	A Ped or Pedigree object.
avail	A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
affected	A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).
affstatus	Affection status to search for.

Details

When used within shrink(), this function is called with the first affected indicator, if the affected item in the Pedigree is a matrix of multiple affected indicators.

If **avail** or **affected** is null, then the function will use the corresponding Ped accessor.

Value

A list is returned with the following components

- ped The new Ped object
- newAvail Vector of availability status of trimmed individuals
- idTrimmed Vector of IDs of trimmed individuals
- isTrimmed logical value indicating whether Ped object has been trimmed
- bit_size Bit size of the trimmed Ped

See Also

shrink()

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
find_avail_affected(pedi, affstatus = 1)
```

---

find_avail_noninform *Find uninformative but available subject*

Description

Finds subjects from among available non-parents with all affection equal to 0.

Usage

```
## S4 method for signature 'Ped'
find_avail_noninform(obj, avail = NULL, affected = NULL)

## S4 method for signature 'Pedigree'
find_avail_noninform(obj, avail = NULL, affected = NULL)
```

Arguments

obj	A Ped or Pedigree object.
avail	A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
affected	A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).

Details

Identify subjects to remove from a Pedigree who are available but non-informative (unaffected). This is the second step to remove subjects in shrink() if the Pedigree does not meet the desired bit size.

If **avail** or **affected** is null, then the function will use the corresponding Ped accessor.

Value

Vector of subject ids who can be removed by having lowest informativeness.

See Also

shrink()

Examples

```
data(sampleped)
pedi <- Pedigree(sampleped)
find_avail_noninform(pedi)
```

---

find_unavailable *Find unavailable subjects in a Pedigree*

Description

Find the identifiers of subjects in a Pedigree iteratively, as anyone who is not available and does not have an available descendant by successively removing unavailable terminal nodes.

Usage

```
## S4 method for signature 'Ped'
find_unavailable(obj, avail = NULL)

## S4 method for signature 'Pedigree'
find_unavailable(obj, avail = NULL)
```

Arguments

obj	A Ped or Pedigree object.
avail	A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).

Details

If **avail** is null, then the function will use the corresponding Ped accessor.

Originally written as pedTrim by Steve Iturria, modified by Dan Schaid 2007, and now split into the two separate functions: `find_unavailable()`, and `trim()` to do the tasks separately. `find_unavailable()` calls `exclude_stray_marryin()` to find stray available marry-ins who are isolated after trimming their unavailable offspring, and `exclude_unavail_founders()`. If the subject ids are character, make sure none of the characters in the ids is a colon (":"), which is a special character used to concatenate and split subjects within the utility. The `trim()` functions is now replaced by the `subset()` function.

Value

Returns a vector of subject ids for who can be removed.

Side Effects

Relation matrix from subsetting is trimmed of any special relations that include the subjects to trim.

See Also

`shrink()`

Examples

```
data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == "1",])
find_unavailable(ped1)
```

`fix_parents` *Fix parents relationship and gender*

Description

Fix the sex of parents, add parents that are missing from the data. Can be used with a dataframe or a vector of the different individuals informations.

Usage

```
## S4 method for signature 'character'
fix_parents(obj, dadid, momid, sex, famid = NULL, missid = NA_character_)

## S4 method for signature 'data.frame'
fix_parents(obj, del_parents = NULL, filter = NULL, missid = NA_character_)
```

Arguments

`obj`	A data.frame or a vector of the individuals identifiers. If a dataframe is given it must contain the columns `id`, `dadid`, `momid`, `sex` and `famid` (optional).
`dadid`	A vector containing for each subject, the identifiers of the biologicals fathers.
`momid`	A vector containing for each subject, the identifiers of the biologicals mothers.
`sex`	A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: `male` < `female` < `unknown` The following values are recognized: • "male": "m", "male", "man", `1` • "female": "f", "female", "woman", `2` • "unknown": "unknown", `3`
`famid`	A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
`missid`	A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to `NA_character_`.
`del_parents`	Boolean defining if missing parents needs to be deleted or fixed. If `one` then if one of the parent is missing, both are removed, if `both` then if both parents are missing, both are removed. If `NULL` then no parent is removed and the missing parents are added as new rows.
`filter`	Filtering column containing `0` or `1` for the rows to kept before proceeding.

Details

First look to add parents whose ids are given in momid/dadid. Second, fix sex of parents. Last look to add second parent for children for whom only one parent id is given. If a **famid** vector is given the family id will be added to the ids of all individuals (`id`, `dadid`, `momid`) separated by an underscore before proceeding.

Special case for dataframe:

Check for presence of both parents id in the **id** field. If not both presence behaviour depend of **delete** parameter

- If TRUE then use fix_parents function and merge back the other fields in the dataframe then set availability to 0 for non available parents.
- If FALSE then delete the id of missing parents

Value

A data.frame with id, dadid, momid, sex as columns with the relationships fixed.

Author(s)

Jason Sinnwell

Examples

```
test1char <- data.frame(
    id = paste('fam', 101:111, sep = ''),
    sex = c('male', 'female')[c(1, 2, 1, 2, 1, 1, 2, 2, 1, 2, 1)],
    father = c(
        0, 0, 'fam101', 'fam101', 'fam101', 0, 0,
        'fam106', 'fam106', 'fam106', 'fam109'
    ),
    mother = c(
        0, 0, 'fam102', 'fam102', 'fam102', 0, 0,
        'fam107', 'fam107', 'fam107', 'fam112'
    )
)
test1newmom <- with(test1char, fix_parents(id, father, mother,
    sex,
    missid = NA_character_
))
Pedigree(test1newmom)
```

---

generate_colors *Process the filling and border colors based on affection and availability*

Description

Perform transformation uppon a dataframe given to compute the colors for the filling and the border of the individuals based on the affection and availability status.

Usage

```
## S4 method for signature 'character'
generate_colors(
  obj,
  avail,
  mods_aff = NULL,
  is_num = FALSE,
  keep_full_scale = FALSE,
  colors_aff = c("yellow2", "red"),
  colors_unaff = c("white", "steelblue4"),
  colors_avail = c("green", "black"),
  colors_na = "grey"
)

## S4 method for signature 'numeric'
generate_colors(
  obj,
  avail,
  threshold = 0.5,
  sup_thres_aff = TRUE,
  is_num = TRUE,
  keep_full_scale = FALSE,
  breaks = 3,
  colors_aff = c("yellow2", "red"),
  colors_unaff = c("white", "steelblue4"),
  colors_avail = c("green", "black"),
  colors_na = "grey"
)

## S4 method for signature 'Pedigree'
generate_colors(
  obj,
  col_aff = "affection",
  add_to_scale = TRUE,
  col_avail = "avail",
  is_num = NULL,
  mods_aff = TRUE,
  threshold = 0.5,
  sup_thres_aff = TRUE,
  keep_full_scale = FALSE,
  breaks = 3,
  colors_aff = c("yellow2", "red"),
  colors_unaff = c("white", "steelblue4"),
  colors_avail = c("green", "black"),
  colors_na = "grey",
  reset = TRUE
)
```

Arguments

obj	A Pedigree object or a vector containing the affection status for each individuals. The affection status can be numeric or a character.
avail	A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
mods_aff	Vector of modality to consider as affected in the case where the values is a factor.
is_num	Boolean defining if the values need to be considered as numeric.
keep_full_scale	Boolean defining if the affection values need to be set as a scale. If values is numeric the filling scale will be calculated based on the values and the number of breaks given. If values isn't numeric then each levels will get it's own color
colors_aff	Set of increasing colors to use for the filling of the affected individuls.
colors_unaff	Set of increasing colors to use for the filling of the unaffected individuls.
colors_avail	Set of 2 colors to use for the box's border of an individual. The first color will be used for available individual (avail == 1) and the second for the unavailable individual (avail == 0).
colors_na	Color to use for individuals with no informations.
threshold	Numeric value separating the affected and healthy subject in the case where the values is numeric.
sup_thres_aff	Boolean defining if the affected individual are above the threshold or not. If TRUE, the individuals will be considered affected if the value of values is stricly above the threshold. If FALSE, the individuals will be considered affected if the value is stricly under the threshold.
breaks	Number of breaks to use when using full scale with numeric values. The same number of breaks will be done for values from affected individuals and unaffected individuals.
col_aff	A character vector with the name of the column to be used for the affection status.
add_to_scale	Boolean defining if the scales need to be added to the existing scales or if they need to replace the existing scales.
col_avail	A character vector with the name of the column to be used for the availability status.
reset	If TRUE the scale of the specified column will be reset if already present.

Details

The colors will be set using the generate_fill()] and the generate_border() functions respectively for the filling and the border.

Value

When used with a vector:

A list of two elements

- The list containing the filling colors processed and their description
- The list containing the border colors processed and their description

When used with a Pedigree object:

The Pedigree object with the `affected` and `avail` columns processed accordingly as well as the `scales` slot updated.

Examples

```
generate_colors(
    c("A", "B", "A", "B", NA, "A", "B", "A", "B", NA),
    c(1, 0, 1, 0, NA, 1, 0, 1, 0, NA),
    mods_aff = "A"
)

generate_colors(
    c(10, 0, 5, 7, NA, 6, 2, 1, 3, NA),
    c(1, 0, 1, 0, NA, 1, 0, 1, 0, NA),
    threshold = 3, keep_full_scale = TRUE
)
data("sampleped")
pedi <- Pedigree(sampleped)
pedi <- generate_colors(pedi, "affection", add_to_scale=FALSE)
scales(pedi)
```

Hints-class *Hints object*

Description

The hints are used to specify the order of the individuals in the pedigree and to specify the order of the spouses.

Constructor ::

You either need to provide **horder** or **spouse** in the dedicated parameters (together or separately), or inside a list.

Usage

```
Hints(horder, spouse)

## S4 method for signature 'list,missing_OR_NULL'
Hints(horder, spouse)

## S4 method for signature 'numeric,data.frame'
Hints(horder, spouse)

## S4 method for signature 'missing_OR_NULL,data.frame'
```

```
Hints(horder, spouse)

## S4 method for signature 'numeric,missing_OR_NULL'
Hints(horder, spouse)
```

Arguments

horder	A named numeric vector with one element per subject in the Pedigree. It determines the relative horizontal order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters). The names of the vector should be the individual identifiers.
spouse	A data.frame with one row per hinted marriage, usually only a few marriages in a pedigree will need an added hint, for instance reverse the plot order of a husband/wife pair. Each row contains the id of the left spouse (i.e. idl), the id of the right hand spouse (i.e. idr), and the anchor (i.e : anchor : 1 = left, 2 = right, 0 = either). Children will preferentially appear under the parents of the anchored spouse.

Value

A Hints object.

Slots

horder A numeric named vector with one element per subject in the Pedigree. It determines the relative horizontal order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters).

spouse A data.frame with one row per hinted marriage, usually only a few marriages in a Pedigree will need an added hint, for instance reverse the plot order of a husband/wife pair. Each row contains the identifiers of the left spouse, the right hand spouse, and the anchor (i.e : 1 = left, 2 = right, 0 = either).

Accessors

- horder(x) : Get the horder vector
- horder(x) <- value : Set the horder vector
- spouse(x) : Get the spouse data.frame
- spouse(x) <- value : Set the spouse data.frame

Generics

- as.list(x): Convert a Hints object to a list
- subset(x, i, keep = TRUE): Subset a Hints object based on the individuals identifiers given.
  - i : A vector of individuals identifiers to keep.
  - keep : A logical value indicating if the individuals should be kept or deleted.

See Also

Pedigree()

Examples

```
Hints(
    list(
        horder = c("1" = 1, "2" = 2, "3" = 3),
        spouse = data.frame(
            idl = c("1", "2"),
            idr = c("2", "3"),
            anchor = c(1, 2)
        )
    )
)

Hints(
    horder = c("1" = 1, "2" = 2, "3" = 3),
    spouse = data.frame(
        idl = c("1", "2"),
        idr = c("2", "3"),
        anchor = c(1, 2)
    )
)

Hints(
    horder = c("1" = 1, "2" = 2, "3" = 3),
    spouse = data.frame(
        idl = c("1", "2"),
        idr = c("2", "3"),
        anchor = c(1, 2)
    )
)

Hints(
    horder = c("1" = 1, "2" = 2, "3" = 3)
)
```

---

ibd_matrix *IBD matrix*

Description

Transform identity by descent (IBD) matrix data from the form produced by external programs such as SOLAR into the compact form used by the coxme and lmekin routines.

Usage

```
ibd_matrix(id1, id2, ibd, idmap, diagonal)
```

Arguments

id1	A character vector with the id of the first individuals of each pairs or a matrix or data frame with 3 columns: id1, id2, and ibd
id2	A character vector with the id of the second individuals of each pairs
ibd	the IBD value for that pair
idmap	an optional 2 column matrix or data frame whose first element is the internal value (as found in id1 and id2, and whose second element will be used for the dimnames of the result
diagonal	optional value for the diagonal element. If present, any missing diagonal elements in the input data will be set to this value.

Details

The IBD matrix for a set of n subjects will be an n by n symmetric matrix whose i,j element is the contains, for some given genetic location, a 0/1 indicator of whether 0, 1/2 or 2/2 of the alleles for i and j are identical by descent. Fractional values occur if the IBD fraction must be imputed. The diagonal will be 1. Since a large fraction of the values will be zero, programs such as Solar return a data set containing only the non-zero elements. As well, Solar will have renumbered the subjects as seq_len(n) in such a way that families are grouped together in the matrix; a separate index file contains the mapping between this new id and the original one. The final matrix should be labeled with the original identifiers.

Value

a sparse matrix of class dsCMatrix. This is the same form used for kinship matrices.

See Also

kinship()

Examples

```
df <- data.frame(
    id1 = c("1", "2", "1"),
    id2 = c("2", "3", "4"),
    ibd = c(0.5, 0.16, 0.27)
)
ibd_matrix(df$id1, df$id2, df$ibd, diagonal = 2)
```

---

is_informative *Find informative individuals*

Description

Select the ids of the informative individuals.

Usage

```
## S4 method for signature 'character_OR_integer'
is_informative(obj, avail, affected, informative = "AvAf")

## S4 method for signature 'Ped'
is_informative(obj, informative = "AvAf", reset = FALSE)

## S4 method for signature 'Pedigree'
is_informative(obj, col_aff = NULL, informative = "AvAf", reset = FALSE)
```

Arguments

obj	A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
avail	A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
affected	A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).
informative	Informative individuals selection can take 5 values: • 'AvAf' (available and affected), • 'AvOrAf' (available or affected), • 'Av' (available only), • 'Af' (affected only), • 'All' (all individuals) • A numeric/character vector of individuals id • A boolean
reset	If TRUE, the isinf slot is reset
col_aff	A character vector with the name of the column to be used for the affection status.

Details

Depending on the **informative** parameter, the function will extract the ids of the informative individuals. In the case of a numeric vector, the function will return the same vector. In the case of a boolean, the function will return the ids of the individuals if TRUE, NA otherwise. In the case of a string, the function will return the ids of the corresponding informative individuals based on the avail and affected columns.

Value

When obj is a vector:

A vector of individuals informative identifiers.

When obj is a Pedigree:

The Pedigree object with its isinf slot updated.

Examples

```
is_informative(c("A", "B", "C", "D", "E"), informative = c("A", "B"))
is_informative(c("A", "B", "C", "D", "E"), informative = c(1, 2))
is_informative(c("A", "B", "C", "D", "E"), informative = c("A", "B"))
is_informative(c("A", "B", "C", "D", "E"), avail = c(1, 0, 0, 1, 1),
    affected = c(0, 1, 0, 1, 1), informative = "AvAf")
is_informative(c("A", "B", "C", "D", "E"), avail = c(1, 0, 0, 1, 1),
    affected = c(0, 1, 0, 1, 1), informative = "AvOrAf")
is_informative(c("A", "B", "C", "D", "E"),
    informative = c(TRUE, FALSE, TRUE, FALSE, TRUE))

data("sampleped")
pedi <- ped(Pedigree(sampleped))
pedi <- is_informative(pedi, informative = "Av")
isinf(pedi)

data("sampleped")
pedi <- Pedigree(sampleped)
pedi <- is_informative(pedi, col_aff = "affection")
isinf(ped(pedi))
```

---

is_parent *Are individuals parents*

Description

Check which individuals are parents.

Usage

```
## S4 method for signature 'character_OR_integer'
is_parent(obj, dadid, momid, missid = NA_character_)

## S4 method for signature 'Ped'
is_parent(obj, missid = NA_character_)
```

Arguments

obj	A vector of each subjects identifiers or a Ped object
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
missid	A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

Value

A vector of boolean of the same size as **obj** with TRUE if the individual is a parent and FALSE otherwise

Examples

```
is_parent(c("1", "2", "3", "4"), c("3", "3", NA, NA), c("4", "4", NA, NA))

data(sampleped)
pedi <- Pedigree(sampleped)
is_parent(ped(pedi))
```

---

kindepth *Individual's depth in a pedigree*

Description

Computes the depth of each subject in the Pedigree.

Usage

```
## S4 method for signature 'character_OR_integer'
kindepth(obj, dadid, momid, align_parents = FALSE, force = FALSE)

## S4 method for signature 'Pedigree'
kindepth(obj, align_parents = FALSE, force = FALSE)

## S4 method for signature 'Ped'
kindepth(obj, align_parents = FALSE, force = FALSE)
```

Arguments

obj	A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
align_parents	If align_parents = TRUE, go one step further and try to make both parents of each child have the same depth. (This is not always possible). It helps the drawing program by lining up pedigrees that 'join in the middle' via a marriage.
force	If force = TRUE, the function will return the depth minus min(depth) if depth reach a state with no founders is not possible.

Details

Mark each person as to their depth in a Pedigree; 0 for a founder, otherwise :

$$depth = 1 + \max(fatherDepth, motherDepth)$$

In the case of an inbred Pedigree a perfect alignment may not exist.

Value

An integer vector containing the depth for each subject

Author(s)

Terry Therneau, updated by Louis Le Nezet

See Also

align()

Examples

```
kindepth(
    c("A", "B", "C", "D", "E"),
    c("C", "D", "0", "0", "0"),
    c("E", "E", "0", "0", "0")
)
data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == "1",])
kindepth(ped1)
```

---

kinship *Kinship matrix*

Description

Compute the kinship matrix for a set of related autosomal subjects. The function is generic, and can accept a Pedigree, a Ped or a vector as the first argument.

Usage

```
## S4 method for signature 'Ped'
kinship(obj, chrtype = "autosome")

## S4 method for signature 'character'
kinship(obj, dadid, momid, sex, chrtype = "autosome")

## S4 method for signature 'Pedigree'
kinship(obj, chrtype = "autosome")
```

Arguments

obj	A Pedigree or Ped object or a vector of subject identifiers.
chrtype	chromosome type. The currently supported types are 'autosome' and 'X'.
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.

`sex` A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: `male` < `female` < `unknown`

The following values are recognized:

- "male": "m", "male", "man", `1`
- "female": "f", "female", "woman", `2`
- "unknown": "unknown", `3`

Details

The function will usually be called with a Pedigree. The call with a Ped or a vector is provided for backwards compatibility with an earlier release of the library that was less capable. Note that when using with a Ped or a vector, any information on twins is not available to the function.

Warning: This function does not work with adopted individuals. If you have adopted individuals in your pedigree, you should remove them before calling this function.

When called with a Pedigree, the routine will create a block-diagonal-symmetric sparse matrix object of class `dsCMatrix`. Since the `[i, j]` value of the result is 0 for any two unrelated individuals i and j and a `Matrix` utilizes sparse representation, the resulting object is often orders of magnitude smaller than an ordinary matrix.

Two genes G1 and G2 are identical by descent (IBD) if they are both physical copies of the same ancestral gene; two genes are identical by state if they represent the same allele. So the brown eye gene that I inherited from my mother is IBD with hers; the same gene in an unrelated individual is not.

The kinship coefficient between two subjects is the probability that a randomly selected allele from a locus will be IBD between them. It is obviously 0 between unrelated individuals. For an autosomal site and no inbreeding it will be 0.5 for an individual with themselves, .25 between mother and child, .125 between an uncle and neice, etc.

The computation is based on a recursive algorithm described in Lange, which assumes that the founder alleles are all independent.

Value

When obj is a vector:

A matrix of kinship coefficients.

When obj is a Pedigree:

A matrix of kinship coefficients ordered by families present in the Pedigree object.

References

K Lange, Mathematical and Statistical Methods for Genetic Analysis, Springer-Verlag, New York, 1997.

See Also

`make_famid()`, `kindepth()`

Examples

```
kinship(c("A", "B", "C", "D", "E"), c("C", "D", "0", "0", "0"),
    c("E", "E", "0", "0", "0"), sex = c(1, 2, 1, 2, 1))
kinship(c("A", "B", "C", "D", "E"), c("C", "D", "0", "0", "0"),
    c("E", "E", "0", "0", "0"), sex = c(1, 2, 1, 2, 1),
    chrtype = "x"
)

data(sampleped)
pedi <- Pedigree(sampleped[-16])
kinship(pedi)
```

---

make_famid *Compute family id*

Description

Construct a family identifier from pedigree information

Usage

```
## S4 method for signature 'character'
make_famid(obj, dadid, momid)

## S4 method for signature 'Pedigree'
make_famid(obj)
```

Arguments

obj	A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.

Details

Create a vector of length n, giving the family 'tree' number of each subject. If the Pedigree is totally connected, then everyone will end up in tree 1, otherwise the tree numbers represent the disconnected subfamilies. Singleton subjects give a zero for family number.

Value

When used with a character vector:

An integer vector giving family groupings

When used with a Pedigree object:

An updated Pedigree object with the family id added and with all ids updated

See Also

kinship()

Examples

```
make_famid(
    c("A", "B", "C", "D", "E", "F"),
    c("C", "D", "0", "0", "0", "0"),
    c("E", "E", "0", "0", "0", "0")
)

data(sampleped)
ped1 <- Pedigree(sampleped[,-1])
make_famid(ped1)
```

---

min_dist_inf *Minimum distance to the informative individuals*

Description

Compute the minimum distance between the informative individuals and all the others. This distance is a transformation of the maximum kinship degree between the informative individuals and all the others. This transformation is done by taking the log2 of the inverse of the maximum kinship degree.

$minDist = log2(1/\max(kinship))$

Therefore, the minimum distance is 1 when the maximum kinship is 0.5 (i.e. same individual) and is infinite when the maximum kinship is 0 (i.e. not related).

For siblings, the kinship value is 0.25 and the minimum distance is 2. Each time the kinship degree is divided by 2, the minimum distance is increased by 1.

Usage

```
## S4 method for signature 'character'
min_dist_inf(obj, dadid, momid, sex, id_inf)

## S4 method for signature 'Pedigree'
min_dist_inf(obj, reset = FALSE, ...)

## S4 method for signature 'Ped'
min_dist_inf(obj, reset = FALSE)
```

Arguments

obj	A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
...	Additional arguments

dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
sex	A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: male < female < unknown The following values are recognized: • "male": "m", "male", "man", 1 • "female": "f", "female", "woman", 2 • "unknown": "unknown", 3
id_inf	An identifiers vector of informative individuals.
reset	If TRUE, the kin and if isinf columns is reset

Value

When obj is a vector:

A vector of the minimum distance between the informative individuals and all the others corresponding to the order of the individuals in the obj vector.

When obj is a Pedigree:

The Pedigree object with a new slot named 'kin' containing the minimum distance between each individuals and the informative individuals. The isinf slot is also updated with the informative individuals.

See Also

kinship()

Examples

```
min_dist_inf(
    c("A", "B", "C", "D", "E"),
    c("C", "D", "0", "0", "0"),
    c("E", "E", "0", "0", "0"),
    sex = c(1, 2, 1, 2, 1),
    id_inf = c("D", "E")
)

data(sampleped)
pedi <- is_informative(
    Pedigree(sampleped),
    informative = "AvAf", col_aff = "affection"
)
kin(ped(min_dist_inf(pedi, col_aff = "affection")))
```

`minnbreast` *Minnesota Breast Cancer Study*

Description

Data from the Minnesota Breast Cancer Family Study. This contains extended pedigrees from 426 families, each identified by a single proband in 1945-1952, with follow up for incident breast cancer.

Usage

```
data(minnbreast)
```

Format

A data frame with 28081 observations, one line per subject, on the following 14 variables.

- `id` : Subject identifier
- `proband` : If 1, this subject is one of the original 426 probands
- `fatherid` : Identifier of the father, if the father is part of the data set; zero otherwise
- `motherid` : Identifier of the mother, if the mother is part of the data set; zero otherwise
- `famid` : Family identifier
- `endage` : Age at last follow-up or incident cancer
- `cancer` : `1` = breast cancer (females) or prostate cancer (males), `0` = censored
- `yob` : Year of birth
- `education` : Amount of education: 1-8 years, 9-12 years, high school graduate, vocational education beyond high school, some college but did not graduate, college graduate, post-graduate education, refused to answer on the questionnaire
- `marstat` : Marital status: married, living with someone in a marriage-like relationship, separated or divorced, widowed, never married, refused to answer the questionaire
- `everpreg` : Ever pregnant at the time of baseline survey
- `parity` : Number of births
- `nbreast` : Number of breast biopsies
- `sex` : `M` or `F`
- `bcpc` : Part of one of the families in the breast / prostate cancer substudy: `0` = no, `1` = yes. Note that subjects who were recruited to the overall study after the date of the BP substudy are coded as zero.

Details

The original study was conducted by Dr. Elving Anderson at the Dight Institute for Human Genetics at the University of Minnesota. From 1944 to 1952, 544 sequential breast cancer cases seen at the University Hospital were enrolled, and information gathered on parents, siblings, offspring, aunts / uncles, and grandparents with the goal of understanding possible familial aspects of brest cancer. In 1991 the study was resurrected by Dr Tom Sellers.

Of the original 544 he excluded 58 prevalent cases, along with another 19 who had less than 2 living relatives at the time of Dr Anderson's survey. Of the remaining 462 families 10 had no living members, 23 could not be located and 8 refused, leaving 426 families on whom updated pedigrees were obtained.

This gave a study with 13351 males and 12699 females (5183 marry-ins). Primary questions were the relationship of early life exposures, breast density, and pharmacogenomics on incident breast cancer risk. For a subset of the families data was gathered on prostate cancer risk for male subjects via questionnaires sent to men over 40. Other than this, data items other than parentage are limited to the female subjects. In 2003 a second phase of the study was instituted. The pedigrees were further extended to the numbers found in this data set, and further data gathered by questionnaire.

References

Epidemiologic and genetic follow-up study of 544 Minnesota breast cancer families: design and methods. Sellers TA, Anderson VE, Potter JD, Bartow SA, Chen PL, Everson L, King RA, Kuni CC, Kushi LH, McGovern PG, et al. Genetic Epidemiology, 1995; 12(4):417-29.

Evaluation of familial clustering of breast and prostate cancer in the Minnesota Breast Cancer Family Study. Grabrick DM, Cerhan JR, Vierkant RA, Therneau TM, Cheville JC, Tindall DJ, Sellers TA. Cancer Detect Prev. 2003; 27(1):30-6.

Risk of breast cancer with oral contraceptive use in women with a family history of breast cancer. Grabrick DM, Hartmann LC, Cerhan JR, Vierkant RA, Therneau TM, Vachon CM, Olson JE, Couch FJ, Anderson KE, Pankratz VS, Sellers TA. JAMA. 2000; 284(14):1791-8.

Examples

```
data(minnbreast)
breastped <- Pedigree(minnbreast,
    cols_ren_ped = list(
        "dadid" = "fatherid", "momid" = "motherid"
    ), missid = "0", col_aff = "cancer"
)
summary(breastped)
scales(breastped)
#plot family 8, proband is solid, slash for cancers
if (interactive()) {
    plot(breastped[famid(ped(breastped)) == "8"], aff_mark = TRUE)
}
```

norm_ped *Normalise a Ped object dataframe*

Description

Normalise dataframe for a Ped object

Usage

```
norm_ped(
  ped_df,
  na_strings = c("NA", ""),
  missid = c(NA_character_, "0"),
  try_num = FALSE,
  cols_used_del = FALSE,
  date_pattern = "%Y-%m-%d"
)
```

Arguments

ped_df

A data.frame with the individuals informations. The minimum columns required are:

- `id` individual identifiers
- `dadid` biological fathers identifiers
- `momid` biological mothers identifiers
- `sex` of the individual

The `famid` column, if provided, will be merged to the *ids* field separated by an underscore using the `upd_famid()` function.

The following columns are also recognize and will be transformed with the `vect_to_binary()` function:

- `deceased` status -> is the individual dead
- `avail` status -> is the individual available
- `evaluated` status -> has the individual a documented evaluation
- `consultand` status -> is the individual the consultand
- `proband` status -> is the individual the proband
- `carrier` status -> is the individual a carrier
- `asymptomatic` status -> is the individual asymptomatic
- `adopted` status -> is the individual adopted

The values recognized for those columns are `1` or `0`, `TRUE` or `FALSE`.

The `fertility` column will be transformed to a factor using the `fertility_to_factor()` function. `infertile_choice_na`, `infertile`, `fertile`

The `miscarriage` column will be transformed to a using the `miscarriage_to_factor()` function. `SAB`, `TOP`, `ECT`, `FALSE`

The `dateofbirth` and `dateofdeath` columns will be transformed to a date object using the `char_to_date()` function.

na_strings	Vector of strings to be considered as NA values.
missid	A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.
try_num	Boolean defining if the function should try to convert all the columns to numeric.
cols_used_del	Boolean defining if the columns that will be used should be deleted.
date_pattern	The pattern of the date

Details

Normalise a dataframe and check for columns correspondance to be able to use it as an input to create a Ped object. Multiple test are done and errors are checked.

Will be considered available any individual with no 'NA' values in the available column. Duplicated id will nullify the relationship of the individual. All individuals with errors will be remove from the dataframe and will be transfered to the error dataframe.

A number of checks are done to ensure the dataframe is correct:

On identifiers::

- All ids (id, dadid, momid, famid) are not empty (!= "")
- All id are unique (no duplicated)
- All dadid and momid are unique in the id column (no duplicated)
- id is not the same as dadid or momid
- Either have both parents or none

On sex::

- All sex code are either male, female, or unknown.
- No parents are infertile or aborted
- All fathers are male
- All mothers are female

Value

A dataframe with different variable correctly standardized and with the errors identified in the error column

See Also

Ped() Ped Pedigree()

Examples

```
df <- data.frame(
    id = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    dadid = c("A", 0, 1, 3, 0, 4, 1, 0, 6, 6),
    momid = c(0, 0, 2, 2, 0, 5, 2, 0, 8, 8),
    famid = c(1, 1, 1, 1, 1, 1, 1, 2, 2, 2),
    sex = c(1, 2, "m", "man", "f", "male", "m", 3, NA, "f"),
    fertility = c(
```

```
      "TRUE", "FALSE", TRUE, FALSE, 1,
      0, "fertile", "infertile", 1, "TRUE"
    ),
    miscarriage = c("TOB", "SAB", NA, FALSE, "ECT", "other", 1, 0, 1, 0),
    deceased = c("TRUE", "FALSE", TRUE, FALSE, 1, 0, 1, 0, 1, 0),
    avail = c("A", "1", 0, NA, 1, 0, 1, 0, 1, 0),
    evalutated = c(
        "TRUE", "FALSE", TRUE, FALSE, 1, 0, NA, "NA", "other", "0"
    ),
    consultand = c(
        "TRUE", "FALSE", TRUE, FALSE, 1, 0, NA, "NA", "other", "0"
    ),
    proband = c("TRUE", "FALSE", TRUE, FALSE, 1, 0, NA, "NA", "other", "0"),
    carrier = c("TRUE", "FALSE", TRUE, FALSE, 1, 0, NA, "NA", "other", "0"),
    asymptomatic = c(
        "TRUE", "FALSE", TRUE, FALSE, 1, 0, NA, "NA", "other", "0"
    ),
    adopted = c("TRUE", "FALSE", TRUE, FALSE, 1, 0, NA, "NA", "other", "0"),
    dateofbirth = c(
         "1978-01-01", "1980-01-01", "1982-01-01", "1984-01-01",
         "1986-01-01", "1988-01-01", "1990-01-01", "1992-01-01",
         "1994-01-01", "1996-01-01"
    ),
    dateofdeath = c(
        "2000-01-01", "2002-01-01", "2004-01-01", NA, "date-not-recognize",
        "NA", "", NA, "2008/01/01", NA
    )
)
tryCatch(
    norm_ped(df),
    error = function(e) print(e)
)
```

`norm_rel` *Normalise a Rel object dataframe*

Description

Normalise a dataframe and check for columns correspondance to be able to use it as an input to create a Ped object.

Usage

```
norm_rel(
  rel_df,
  multi_code = "error",
  na_strings = c("NA", ""),
  missid = c(NA_character_, "0")
)
```

Arguments

rel_df	A data.frame with the special relationships between individuals. See Rel() for more informations. The minimum columns required are id1, id2 and code. The famid column can also be used to specify the family of the individuals. If a matrix is given, the columns needs to be ordered as id1, id2, code and famid. The code values are: • 1 = Monozygotic twin • 2 = Dizygotic twin • 3 = twin of unknown zygosity • 4 = Spouse The value relation code recognized by the function are the one defined by the rel_code_to_factor() function.
multi_code	How to handle multiple relationship codes in the same group If "error", an error is thrown. If "warn", a warning is thrown and the relationship code is set to twins of unknow zigosity. Default is "error".
na_strings	Vector of strings to be considered as NA values.
missid	A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

Details

The famid column, if provided, will be merged to the *ids* field separated by an underscore using the upd_famid() function. The code column will be transformed with the rel_code_to_factor(). Missing relationship for set of twins will be completed using complete_twins(). Multiple test are done and errors are checked.

A number of checks are done to ensure the dataframe is correct:

On identifiers::

- All ids (id1, id2) are not empty (!= "")
- id1 and id2 are not the same

On code:

- All code are recognised as either "MZ twin", "DZ twin", "UZ twin" or "Spouse"

Value

A dataframe with the errors identified

Examples

```
df <- data.frame(
    id1 = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    id2 = c(2, 3, 4, 5, 6, 7, 8, 9, 10, 1),
    code = c("MZ twin", "DZ twin", "UZ twin", "Spouse",
        1, 2, 3, 4, "MzTwin", "sp oUse"),
    famid = c(1, 1, 1, 1, 1, 1, 1, 2, 2, 2)
)
```

```
norm_rel(df)
```

---

num_child *Number of childs*

Description

Compute the number of childs per individual

Usage

```
## S4 method for signature 'character_OR_integer'
num_child(obj, dadid, momid, rel_df = NULL, missid = NA_character_)

## S4 method for signature 'Pedigree'
num_child(obj, reset = FALSE)
```

Arguments

obj	A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
rel_df	A data.frame with the special relationships between individuals. See Rel() for more informations. The minimum columns required are id1, id2 and code. The famid column can also be used to specify the family of the individuals. If a matrix is given, the columns needs to be ordered as id1, id2, code and famid. The code values are: • 1 = Monozygotic twin • 2 = Dizygotic twin • 3 = twin of unknown zygosity • 4 = Spouse The value relation code recognized by the function are the one defined by the rel_code_to_factor() function.
missid	A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.
reset	If TRUE, the num_child_tot, num_child_ind and the num_child_dir columns are reset.

Details

Compute the number of direct child but also the number of indirect child given by the ones related with the linked spouses. If a relation ship dataframe is given, then even if no children is present between 2 spouses, the indirect childs will still be added.

Value

When obj is a vector:

A dataframe with the columns num_child_dir, num_child_ind and num_child_tot giving respectively the direct, indirect and total number of child.

When obj is a Pedigree object:

An updated Pedigree object with the columns num_child_dir, num_child_ind and num_child_tot added to the Pedigree ped slot.

Examples

```
num_child(
    obj = c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10"),
    dadid = c("3", "3", "6", "8", "0", "0", "0", "0", "0", "0"),
    momid = c("4", "5", "7", "9", "0", "0", "0", "0", "0", "0"),
    rel_df = data.frame(
        id1 = "10",
        id2 = "3",
        code = "Spouse"
    )
)

data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == "1",])
ped1 <- num_child(ped1, reset = TRUE)
summary(ped(ped1))
```

---

parent_of *Get parents of individuals*

Description

Get the parents of individuals.

Usage

```
## S4 method for signature 'character_OR_integer'
parent_of(obj, dadid, momid, id2)

## S4 method for signature 'Ped'
parent_of(obj, id2)

## S4 method for signature 'Pedigree'
parent_of(obj, id2)
```

Arguments

obj	A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
id2	A vector of individuals identifiers to get the parents from

Value

A vector of individuals identifiers corresponding to the parents of the individuals in **id2**

Examples

```
data(sampleped)
ped <- Pedigree(sampleped)
parent_of(ped, "1_121")
```

Ped-class *Ped object*

Description

S4 class to represent the identity informations of the individuals in a pedigree.

Constructor ::

You either need to provide a vector of the same size for each slot or a data.frame with the corresponding columns.

The metadata will correspond to the columns that do not correspond to the Ped slots.

Usage

```
## S4 method for signature 'data.frame'
Ped(
  obj,
  cols_used_init = FALSE,
  cols_used_del = FALSE,
  date_pattern = "%Y-%m-%d"
)

## S4 method for signature 'character_OR_integer'
Ped(
  obj,
  dadid,
  momid,
  sex,
  famid = NA,
```

```
  fertility = NA,
  miscarriage = NA,
  deceased = NA,
  avail = NA,
  evaluated = NA,
  consultand = NA,
  proband = NA,
  affected = NA,
  carrier = NA,
  asymptomatic = NA,
  adopted = NA,
  dateofbirth = NA,
  dateofdeath = NA,
  missid = c(NA_character_, "0"),
  useful = NA,
  isinf = NA,
  kin = NA_real_
)
```

Arguments

obj	A character vector with the id of the individuals or a `data.frame` with all the informations in corresponding columns.
cols_used_init	Boolean defining if the columns that will be used should be initialised to NA.
cols_used_del	Boolean defining if the columns that will be used should be deleted.
date_pattern	The pattern of the date
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
sex	A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: `male` < `female` < `unknown` The following values are recognized: • "male": "m", "male", "man", `1` • "female": "f", "female", "woman", `2` • "unknown": "unknown", `3`
famid	A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
fertility	A character, factor or numeric vector corresponding to the fertility status of the individuals. This will be transformed to a factor with the following levels: `infertile_choice_na`, `infertile`, `fertile` The following values are recognized: • "inferile_choice_na" : "infertile_choice", "infertile_na" • "infertile" : "infertile", "steril", `FALSE`, `0` • "fertile" : "fertile", `TRUE`, `1`, `NA`

`miscarriage` A character, factor or numeric vector corresponding to the miscarriage status of the individuals. This will be transformed to a factor with the following levels: `TOP`, `SAB`, `ECT`, `FALSE` The following values are recognized:

- "SAB" : "spontaneous", "spontaenous abortion"
- "TOP" : "termination", "terminated", "termination of pregnancy"
- "ECT" : "ectopic", "ectopic pregnancy"
- FALSE : `0`, `FALSE`, "no", `NA`

`deceased` A logical vector with the death status of the individuals (i.e. `FALSE` = alive, `TRUE` = dead, `NA` = unknown).

`avail` A logical vector with the availability status of the individuals (i.e. `FALSE` = not available, `TRUE` = available, `NA` = unknown).

`evaluated` A logical vector with the evaluation status of the individuals. (i.e. `FALSE` = documented evaluation not available, `TRUE` = documented evaluation available).

`consultand` A logical vector with the consultand status of the individuals. A consultand being an individual seeking genetic counseling/testing (i.e. `FALSE` = not a consultand, `TRUE` = consultand).

`proband` A logical vector with the proband status of the individuals. A proband being an affected family member coming to medical attention independent of other family members. (i.e. `FALSE` = not a proband, `TRUE` = proband).

`affected` A logical vector with the affection status of the individuals (i.e. `FALSE` = unaffected, `TRUE` = affected, `NA` = unknown).

`carrier` A logical vector with the carrier status of the individuals. A carrier being an individual who has the genetic trait but who is not likely to manifest the disease regardless of inheritance pattern (i.e. `FALSE` = not carrier, `TRUE` = carrier, `NA` = unknown).

`asymptomatic` A logical vector with the asymptomatic status of the individuals. An asymptomatic individual being an individual clinically unaffected at this time but could later exhibit symptoms. (i.e. `FALSE` = not asymptomatic, `TRUE` = asymptomatic, `NA` = unknown).

`adopted` A logical vector with the adopted status of the individuals. (i.e. `FALSE` = not adopted, `TRUE` = adopted, `NA` = unknown).

`dateofbirth` A character vector with the date of birth of the individuals.

`dateofdeath` A character vector with the date of death of the individuals.

`missid` A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to `NA_character_`.

`useful` A logical vector with the usefulness status of the individuals (i.e. `FALSE` = not useful, `TRUE` = useful).

`isinf` A logical vector indicating if the individual is informative or not (i.e. `FALSE` = not informative, `TRUE` = informative).

`kin` A numeric vector with minimal kinship value between the individuals and the informative individuals.

Details

The minimal needed informations are `id`, `dadid`, `momid` and `sex`. The other slots are used to store recognized informations. Additional columns can be added to the Ped object and will be stored in the `elementMetadata` slot of the Ped object.

Value

A Ped object.

Slots

`id` A character vector with the id of the individuals.

`dadid` A character vector with the id of the father of the individuals.

`momid` A character vector with the id of the mother of the individuals.

`famid` A character vector with the family identifiers of the individuals (optional).

`sex` An ordered factor vector for the sex of the individuals (i.e. `male` < `female` < `unknown`).

`fertility` A factor vector with the fertility status of the individuals (optional). (i.e. `infertile_choice_na` = no children by choice or unknown reason, `infertile` = individual is inferile, `fertile` = individual is fertile).

`miscarriage` A factor vector with the miscarriage status of the individuals (optional). (i.e. `TOP` = Termination of Pregnancy, `SAB` = Spontaneous Abortion, `ECT` = Ectopic Pregnancy, `FALSE` = no miscarriage).

`deceased` A logical vector with the death status of the individuals (optional). (i.e. `FALSE` = alive, `TRUE` = dead, `NA` = unknown).

`avail` A logical vector with the availability status of the individuals (optional). (i.e. `FALSE` = not available, `TRUE` = available, `NA` = unknown).

`evaluated` A logical vector with the evaluation status of the individuals (optional). (i.e. `FALSE` = documented evaluation not available, `TRUE` = documented evaluation available).

`consultand` A logical vector with the consultand status of the individuals (optional). A consultand being an individual seeking genetic counseling/testing (i.e. `FALSE` = not a consultand, `TRUE` = consultand).

`proband` A logical vector with the proband status of the individuals (optional). A proband being an affected family member coming to medical attention independent of other family members. (i.e. `FALSE` = not a proband, `TRUE` = proband).

`affected` A logical vector with the affection status of the individuals (optional). (i.e. `FALSE` = not affected, `TRUE` = affected, `NA` = unknown).

`carrier` A logical vector with the carrier status of the individuals (optional). A carrier being an individual who has the genetic trait but who is not likely to manifest the disease regardless of inheritance pattern (i.e. `FALSE` = not carrier, `TRUE` = carrier, `NA` = unknown).

`asymptomatic` A logical vector with the asymptomatic status of the individuals (optional). An asymptomatic individual being an individual clinically unaffected at this time but could later exhibit symptoms. (i.e. `FALSE` = not asymptomatic, `TRUE` = asymptomatic, `NA` = unknown).

`adopted` A logical vector with the adopted status of the individuals (optional). (i.e. `FALSE` = not adopted, `TRUE` = adopted, `NA` = unknown).

`dateofbirth` A date vector with the birth date of the individuals (optional).

`dateofdeath` A date vector with the death date of the individuals (optional).

`useful` A logical vector with the usefulness status of the individuals (computed). (i.e. `FALSE` = not useful, `TRUE` = useful).

`isinf` A logical vector indicating if the individual is informative or not (computed). (i.e. `FALSE` = not informative, `TRUE` = informative).

`kin` A numeric vector with minimal kinship value between the individuals and the useful individuals (computed).

`num_child_tot` A numeric vector with the total number of children of the individuals (computed).

`num_child_dir` A numeric vector with the number of children of the individuals (computed).

`num_child_ind` A numeric vector with the number of children of the individuals (computed).

`elementMetadata` A DataFrame with the additional metadata columns of the Ped object.

`metadata` Meta informations about the pedigree.

Accessors

For all the following accessors, the `x` parameters is a Ped object. Each getters return a vector of the same length as `x` with the values of the corresponding slot. For each getter, you have a setter with the same name, to be use as `slot(x) <- value`. The `value` parameter is a vector of the same length as `x`, except for the `mcols()` accessors where `value` is a list or a data.frame with each elements with the same length as `x`.

- `id(x)` : Individuals identifiers
- `dadid(x)` : Individuals' father identifiers
- `momid(x)` : Individuals' mother identifiers
- `famid(x)` : Individuals' family identifiers
- `sex(x)` : Individuals' gender
- `fertility(x)` : Individuals' fertility status
- `miscarriage(x)` : Individuals' miscarriage status
- `deceased(x)` : Individuals' death status
- `avail(x)` : Individuals' availability status
- `evaluated(x)` : Individuals' evaluated status
- `consultand(x)` : Individuals' consultand status
- `proband(x)` : Individuals' proband status
- `carrier(x)` : Individuals' carrier status
- `asymptomatic(x)` : Individuals' asymptomatic status

- `adopted(x)` : Individuals' adopted status
- `affected(x)` : Individuals' affection status
- `dateofbirth(x)` : Individuals' birth dates
- `dateofdeath(x)` : Individuals' death dates
- `isinf(x)` : Individuals' informativeness status
- `kin(x)` : Individuals' kinship distance to the informative individuals
- `useful(x)` : Individuals' usefullness status
- `mcols(x)` : Individuals' metadata

Generics

- `summary(x)`: Compute the summary of a Ped object
- `show(x)`: Convert the Ped object to a data.frame and print it with its summary.
- `as.list(x)`: Convert a Ped object to a list with the metadata columns at the end.
- `as.data.frame(x)`: Convert a Ped object to a data.frame with the metadata columns at the end.
- `subset(x, i, del_parents = FALSE, keep = TRUE)`: Subset a Ped object based on the individuals identifiers given.
  - `i` : A vector of individuals identifiers to keep.
  - `del_parents` : A value indicating if the parents of the individuals should be deleted.
  - `keep` : A logical value indicating if the individuals should be kept or deleted.

See Also

`Pedigree()`

Examples

```
data(sampleped)
Ped(sampleped)

Ped(
    obj = c("1", "2", "3", "4", "5", "6"),
    dadid = c("4", "4", "6", "0", "0", "0"),
    momid = c("5", "5", "5", "0", "0", "0"),
    sex = c(1, 2, 3, 1, 2, 1),
    missid = "0"
)
```

ped_server *Create the server logic for the ped_shiny application*

Description

Create the server logic for the ped_shiny application

Usage

```
ped_server(
  input,
  output,
  session,
  precision = 6,
  ind_max_warning = 300,
  ind_max_error = 500
)
```

Arguments

input	The input object from a Shiny app.
output	The output object from a Shiny app.
session	The session object from a Shiny app.
precision	Number of decimal for the position of the boxes in the plot.
ind_max_warning	An integer to define the maximum number of individuals to plot before throwing a warning. If the number of individuals is greater than this value, the user will be asked to confirm the plot.
ind_max_error	An integer to define the maximum number of individuals to plot before throwing an error. If the number of individuals is greater than this value, an error will be thrown and the plot will not be computed.

Value

`shiny::shinyServer()`

Examples

```
if (interactive()) {
    ped_shiny()
}
```

`ped_shiny` *Run Pedixplorer Shiny application*

Description

This function creates a shiny application to manage and visualize pedigree data using the `ped_ui()` and `ped_server()` functions.

Usage

```
ped_shiny(
  port = getOption("shiny.port"),
  host = getOption("shiny.host", "127.0.0.1"),
  precision = 6,
  ind_max_warning = 300,
  ind_max_error = 500
)
```

Arguments

`port`	(optional) Specify port the application should list to.
`host`	(optional) The IPv4 address that the application should listen on.
`precision`	Number of decimal for the position of the boxes in the plot.
`ind_max_warning`	An integer to define the maximum number of individuals to plot before throwing a warning. If the number of individuals is greater than this value, the user will be asked to confirm the plot.
`ind_max_error`	An integer to define the maximum number of individuals to plot before throwing an error. If the number of individuals is greater than this value, an error will be thrown and the plot will not be computed.

Details

The application is composed of several modules:

- Data import
- Data column selection
- Data download
- Family selection
- Health selection
- Informative selection
- Subfamily selection
- Plotting pedigree
- Family information

Value

Running Shiny Application

Examples

```
if (interactive()) {
    ped_shiny()
}
```

`ped_to_legdf` *Create plotting legend data frame from a Pedigree*

Description

Convert a Pedigree to a legend data frame for it to be plotted afterwards with `plot_fromdf()`.

Usage

```
## S4 method for signature 'Pedigree'
ped_to_legdf(
  obj,
  boxh = 1,
  boxw = 1,
  cex = 1,
  adjx = 0,
  adjy = 0,
  lwd = 1,
  precision = 4
)
```

Arguments

`obj`	A Pedigree object
`boxh`	Height of the polygons elements
`boxw`	Width of the polygons elements
`cex`	Character expansion of the text
`adjx`	default=0. Controls the horizontal text adjustment of the labels in the legend.
`adjy`	default=0. Controls the vertical text adjustment of the labels in the legend.
`lwd`	default=1. Controls the bordering line width of the elements in the legend.
`precision`	The number of significatif numbers to round the numbers to.

Details

The data frame contains the following columns:

- `x0`, `y0`, `x1`, `y1`: coordinates of the elements
- `type`: type of the elements
- `fill`: fill color of the elements
- `border`: border color of the elements
- `angle`: angle of the shading of the elements
- `density`: density of the shading of the elements
- `cex`: size of the elements
- `label`: label of the elements
- `tips`: tips of the elements (used for the tooltips)
- `adjx`: horizontal text adjustment of the labels
- `adjy`: vertical text adjustment of the labels

All those columns are used by `plot_fromdf()` to plot the graph.

Value

A list containing the legend data frame and the user coordinates.

Examples

```
data("sampleped")
pedi <- Pedigree(sampleped)
leg_df <- ped_to_legdf(pedi)
summary(leg_df$df)
plot_fromdf(leg_df$df, usr = c(-1,15,0,7))
```

---

ped_to_plotdf *Create plotting data frame from a Pedigree*

Description

Convert a Pedigree to a data frame with all the elements and their characteristic for them to be plotted afterwards with `plot_fromdf()`.

Usage

```
## S4 method for signature 'Pedigree'
ped_to_plotdf(
  obj,
  packed = TRUE,
  width = 6,
  align = c(1.5, 2),
  align_parents = TRUE,
  force = FALSE,
  cex = 1,
  symbolsize = cex,
  pconnect = 0.5,
  branch = 0.6,
  aff_mark = TRUE,
  id_lab = "id",
  label = NULL,
  precision = 4,
  lwd = 1,
  tips = NULL,
  ggplot_gen = FALSE,
  label_dist = c(1, 3, 5),
  label_cex = c(1, 0.7, 1),
  ...
)
```

Arguments

obj	A Pedigree object
...	Other arguments passed to par()
packed	Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
width	For a packed output, the minimum width of the plot, in inches.
align	For a packed Pedigree, align children under parents TRUE, to the extent possible given the page width, or align to to the left margin FALSE. This argument can be a two element vector, giving the alignment parameters, or a logical value. If TRUE, the default is c(1.5, 2), or if numeric the routine alignped4() will be called.
align_parents	If align_parents = TRUE, go one step further and try to make both parents of each child have the same depth. (This is not always possible). It helps the drawing program by lining up pedigrees that 'join in the middle' via a marriage.
force	If force = TRUE, the function will return the depth minus min(depth) if depth reach a state with no founders is not possible.
cex	Character expansion of the text
symbolsize	Size of the symbols
pconnect	When connecting parent to children the program will try to make the connecting line as close to vertical as possible, subject to it lying inside the endpoints of the

	line that connects the children by at least pconnect people. Setting this option to a large number will force the line to connect at the midpoint of the children.
branch	defines how much angle is used to connect various levels of nuclear families.
aff_mark	If TRUE, add a aff_mark to each box corresponding to the value of the affection column for each filling scale.
id_lab	The column name of the id for each individuals.
label	If not NULL, add a label to each box under the id corresponding to the value of the column given.
precision	The number of significatif numbers to round the solution to.
lwd	default=1. Controls the line width of the segments, arcs and polygons.
tips	A character vector of the column names of the data frame to use as tooltips. If NULL, no tooltips are added.
ggplot_gen	If TRUE, the function will use the ggplot2 package to generate the plot.
label_dist	A numeric vector of length 3 giving the distance between the id, date and label text and the bottom of the box. This value is multiplied by the obtained labh value.
label_cex	A numeric vector of length 3 giving the cex of the id, date and label text. This value is multiplied by the cex argument

Details

The data frame contains the following columns:

- x0, y0, x1, y1: coordinates of the elements
- type: type of the elements
- fill: fill color of the elements
- border: border color of the elements
- angle: angle of the shading of the elements
- density: density of the shading of the elements
- cex: size of the elements
- label: label of the elements
- tips: tips of the elements (used for the tooltips)
- adjx: horizontal text adjustment of the labels
- adjy: vertical text adjustment of the labels

All those columns are used by plot_fromdf() to plot the graph.

Value

A list containing the data frame and the user coordinates.

See Also

plot_fromdf() ped_to_legdf()

Examples

```
data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == 1,])
plot_df <- ped_to_plotdf(ped1)
summary(plot_df$df)
plot_fromdf(plot_df$df, usr = plot_df$par_usr$usr,
    boxh = plot_df$par_usr$boxh, boxw = plot_df$par_usr$boxw
)
```

---

ped_ui *Create the user interface for the ped_shiny application*

Description

Create the user interface for the ped_shiny application

Value

`shiny::shinyUI()`

Examples

```
if (interactive()) {
    ped_shiny()
}
```

Pedigree-class *Pedigree object*

Description

A pedigree is a ensemble of individuals linked to each other into a family tree. A Pedigree object store the informations of the individuals and the special relationships between them. It also permit to store the informations needed to plot the pedigree (i.e. scales and hints).

Constructor ::

Main constructor of the package. This constructor help to create a `Pedigree` object from different `data.frame` or a set of vectors.

If any errors are found in the data, the function will return the data.frame with the errors of the Ped object and the Rel object.

Usage

```
Pedigree(obj, ...)

## S4 method for signature 'character_OR_integer'
Pedigree(
  obj,
  dadid,
  momid,
  sex,
  famid = NA,
  fertility = NULL,
  miscarriage = NULL,
  deceased = NULL,
  avail = NULL,
  evaluated = NULL,
  consultand = NULL,
  proband = NULL,
  affections = NULL,
  carrier = NULL,
  asymptomatic = NULL,
  adopted = NULL,
  dateofbirth = NULL,
  dateofdeath = NULL,
  rel_df = NULL,
  missid = c(NA_character_, "0"),
  col_aff = "affection",
  date_pattern = "%Y-%m-%d",
  normalize = TRUE,
  ...
)

## S4 method for signature 'data.frame'
Pedigree(
 obj = data.frame(id = character(), dadid = character(), momid = character(), famid =
   character(), sex = numeric(), fertility = numeric(), miscarriage = numeric(),
   deceased = numeric(), avail = numeric(), evaluated = logical(), consultand =
   logical(), proband = logical(), affection = logical(), carrier = logical(),
  asymptomatic = logical(), adopted = logical(), dateofbirth = character(), dateofdeath
   = character()),
 rel_df = data.frame(id1 = character(), id2 = character(), code = numeric(), famid =
   character()),
 cols_ren_ped = list(id = "indId", dadid = "fatherId", momid = "motherId", famid =
  "family", sex = "gender", fertility = c("sterilisation", "steril"), miscarriage =
  c("miscarriage", "aborted"), deceased = c("status", "dead", "vitalStatus"), avail =
   "available", evaluated = "evaluation", consultand = "consultant", proband =
   "proband", affection = "affected", carrier = "carrier", asymptomatic =
  "presymptomatic", adopted = "adoption", dateofbirth = c("dob", "birth"), dateofdeath
   = c("dod", "death")),
```

```
  cols_ren_rel = list(id1 = "indId1", id2 = "indId2", famid = "family"),
  hints = list(horder = NULL, spouse = NULL),
  normalize = TRUE,
  missid = c(NA_character_, "0"),
  col_aff = "affection",
  date_pattern = "%Y-%m-%d",
  na_strings = c("NA", "N/A", "None", "none", "null", "NULL"),
  ...
)
```

Arguments

obj	A vector of the individuals identifiers or a data.frame with the individuals informations. See Ped() for more informations.
...	Arguments passed on to generate_colors
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
sex	A character, factor or numeric vector corresponding to the gender of the individuals. This will be transformed to an ordered factor with the following levels: male < female < unknown The following values are recognized: • "male": "m", "male", "man", 1 • "female": "f", "female", "woman", 2 • "unknown": "unknown", 3
famid	A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
fertility	A character, factor or numeric vector corresponding to the fertility status of the individuals. This will be transformed to a factor with the following levels: infertile_choice_na, infertile, fertile The following values are recognized: • "inferile_choice_na" : "infertile_choice", "infertile_na" • "infertile" : "infertile", "steril", FALSE, 0 • "fertile" : "fertile", TRUE, 1, NA
miscarriage	A character, factor or numeric vector corresponding to the miscarriage status of the individuals. This will be transformed to a factor with the following levels: TOP, SAB, ECT, FALSE The following values are recognized: • "SAB" : "spontaneous", "spontaenous abortion" • "TOP" : "termination", "terminated", "termination of pregnancy" • "ECT" : "ectopic", "ectopic pregnancy" • FALSE : 0, FALSE, "no", NA
deceased	A logical vector with the death status of the individuals (i.e. FALSE = alive, TRUE = dead, NA = unknown).

`avail` A logical vector with the availability status of the individuals (i.e. `FALSE` = not available, `TRUE` = available, `NA` = unknown).

`evaluated` A logical vector with the evaluation status of the individuals. (i.e. `FALSE` = documented evaluation not available, `TRUE` = documented evaluation available).

`consultand` A logical vector with the consultand status of the individuals. A consultand being an individual seeking genetic counseling/testing (i.e. `FALSE` = not a consultand, `TRUE` = consultand).

`proband` A logical vector with the proband status of the individuals. A proband being an affected family member coming to medical attention independent of other family members. (i.e. `FALSE` = not a proband, `TRUE` = proband).

`affections` A logical vector with the affections status of the individuals (i.e. `FALSE` = unaffected, `TRUE` = affected, `NA` = unknown). Can also be a data.frame with the same length as `obj`. If it is a matrix, it will be converted to a data.frame and the columns will be named after the `col_aff` argument.

`carrier` A logical vector with the carrier status of the individuals. A carrier being an individual who has the genetic trait but who is not likely to manifest the disease regardless of inheritance pattern (i.e. `FALSE` = not carrier, `TRUE` = carrier, `NA` = unknown).

`asymptomatic` A logical vector with the asymptomatic status of the individuals. An asymptomatic individual being an individual clinically unaffected at this time but could later exhibit symptoms. (i.e. `FALSE` = not asymptomatic, `TRUE` = asymptomatic, `NA` = unknown).

`adopted` A logical vector with the adopted status of the individuals. (i.e. `FALSE` = not adopted, `TRUE` = adopted, `NA` = unknown).

`dateofbirth` A character vector with the date of birth of the individuals.

`dateofdeath` A character vector with the date of death of the individuals.

`rel_df` A data.frame with the special relationships between individuals. See `Rel()` for more informations. The minimum columns required are `id1`, `id2` and `code`. The `famid` column can also be used to specify the family of the individuals. If a matrix is given, the columns needs to be ordered as `id1`, `id2`, `code` and `famid`. The code values are:

- `1` = Monozygotic twin
- `2` = Dizygotic twin
- `3` = twin of unknown zygosity
- `4` = Spouse

The value relation code recognized by the function are the one defined by the `rel_code_to_factor()` function.

`missid` A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to `NA_character_`.

`col_aff` A character vector with the name of the column to be used for the affection status.

`date_pattern` The pattern of the date

`normalize` A logical to know if the data should be normalised.

cols_ren_ped	A named list with the columns to rename for the pedigree dataframe. This is useful if you want to use a dataframe with different column names. The names of the list should be the new column names and the values should be the old column names. The default values are to be used with normalize = TRUE.
cols_ren_rel	A named list with the columns to rename for the relationship matrix. This is useful if you want to use a dataframe with different column names. The names of the list should be the new column names and the values should be the old column names.
hints	A Hints object or a named list containing horder and spouse.
na_strings	Vector of strings to be considered as NA values.

Details

If the normalization is set to TRUE, then the data will be standardized using the function norm_ped() and norm_rel().

If a data.frame is given, the columns names needed are as follow:

- id: the individual identifier
- dadid: the identifier of the biological father
- momid: the identifier of the biological mother
- famid: the family identifier of the individual
- sex: the sex of the individual
- fertility: the fertility status of the individual
- miscarriage: the miscarriage status of the individual
- deceased: the death status of the individual
- avail: the availability status of the individual
- evaluated: the evaluation status of the individual
- consultand: the consultand status of the individual
- proband: the proband status of the individual
- affection: the affection status of the individual
- carrier: the carrier status of the individual
- asymptomatic: the asymptomatic status of the individual
- adopted: the adopted status of the individual
- dateofbirth: the date of birth of the individual
- dateofdeath: the date of death of the individual
- ...: other columns that will be stored in the elementMetadata slot

The minimum columns required are :

- id
- dadid
- momid

- sex

The famid column can also be used to specify the family of the individuals and will be merge to the id field separated by an underscore.

The columns deceased, avail, evaluated, consultand, proband, carrier, asymptomatic, adopted will be transformed with the vect_to_binary() function when the normalisation is selected.

The fertility column will be transformed with the fertility_to_factor() function.

The miscarriage column will be transformed with the miscarriage_to_factor() function.

The dateofbirth and dateofdeath columns will be transformed with the char_to_date() function.

If affections is a data.frame, **col_aff** will be overwritten by the column names of the data.frame.

Value

A Pedigree object.

Slots

ped A Ped object for the identity informations. See Ped() for more informations.

rel A Rel object for the special relationships. See Rel() for more informations.

scales A Scales object for the filling and bordering colors used in the plot. See Scales() for more informations.

hints A Hints object for the ordering of the individuals in the plot. See Hints() for more informations.

Accessors

- ped(x, slot) : Get the value of a specific slot of the Ped object
- ped(x) : Get the Ped object
- ped(x, slot) <- value : Set the value of a specific slot of the Ped object Wrapper of slot(ped(x)) <- value
- ped(x) <- value : Set the Ped object
- mcols(x) : Get the metadata of a Pedigree object. This function is a wrapper around mcols(ped(x)).
- mcols(x) <- value : Set the metadata of a Pedigree object. This function is a wrapper around mcols(ped(x)) <- value.
- rel(x, slot) : Get the value of a specific slot of the Rel object
- rel(x) : Get the Rel object
- rel(x, slot) <- value : Set the value of a specific slot of the Rel object Wrapper of slot(rel(x)) <- value
- rel(x) <- value : Set the Rel object

- `scales(x)` : Get the Scales object
- `scales(x) <- value` : Set the Scales object
- `fill(x)` : Get the fill data.frame from the Scales object. Wrapper of `fill(scales(x))`
- `fill(x) <- value` : Set the fill data.frame from the Scales object. Wrapper of `fill(scales(x)) <- value`
- `border(x)` : Get the border data.frame from the Scales object. Wrapper of `border(scales(x))`
- `border(x) <- value` : Set the border data.frame from the Scales object. Wrapper of `border(scales(x)) <- value`
- `hints(x)` : Get the Hints object
- `hints(x) <- value` : Set the Hints object
- `horder(x)` : Get the horder vector from the Hints object. Wrapper of `horder(hints(x))`
- `horder(x) <- value` : Set the horder vector from the Hints object. Wrapper of `horder(hints(x)) <- value`
- `spouse(x)` : Get the spouse data.frame from the Hints object. Wrapper of `spouse(hints(x))`.
- `spouse(x) <- value` : Set the spouse data.frame from the Hints object. Wrapper of `spouse(hints(x)) <- value`.

Generics

- `length(x)`: Get the length of a Pedigree object. Wrapper of `length(ped(x))`.
- `show(x)`: Print the information of the Ped and Rel object inside the Pedigree object.
- `summary(x)`: Compute the summary of the Ped and Rel object inside the Pedigree object.
- `as.list(x)`: Convert a Pedigree object to a list
- `subset(x, i, keep = TRUE)`: Subset a Pedigree object based on the individuals identifiers given.
  - `i` : A vector of individuals identifiers to keep.
  - `del_parents` : A logical value indicating if the parents of the individuals should be deleted.
  - `keep` : A logical value indicating if the individuals should be kept or deleted.
- `x[i, del_parents, keep]`: Subset a Pedigree object based on the individuals identifiers given.

See Also

`Pedigree()` `Ped()` `Rel()` `Scales()` `Hints()`

`Ped()` `Rel()` `Scales()`

Examples

```
Pedigree(
    obj = c("1", "2", "3", "4", "5", "6"),
    dadid = c("4", "4", "6", "0", "0", "0"),
    momid = c("5", "5", "5", "0", "0", "0"),
    sex = c(1, 2, 3, 1, 2, 1),
    avail = c(0, 1, 0, 1, 0, 1),
    affections = matrix(c(
        0, 1, 0, 1, 0, 1,
        1, 1, 1, 1, 1, 1
    ), ncol = 2),
    col_aff = c("aff1", "aff2"),
    missid = "0",
    rel_df = matrix(c(
        "1", "2", 2
    ), ncol = 3, byrow = TRUE),
)

data(sampleped)
Pedigree(sampleped)
```

---

plink_to_pedigree *Import from .fam file or .ped file*

Description

Import a .fam or .ped file and return a Pedigree object

Usage

```
plink_to_pedigree(
  path,
  sep = "\t",
  quote = "'",
  header = FALSE,
  na_values = c("NA", "0")
)
```

Arguments

path	Path to the file
sep	Separator used in the file
quote	Quote used in the file
header	Boolean defining if the file has a header
na_values	A vector of strings that should be considered as NA

Value

A Pedigree object

Examples

```
if (interactive()) {
    write.table(
        data.frame(
            famid = c("1", "1", "1"),
            id = c("A", "B", "C"),
            dadid = c(0, 0, "A"),
            momid = c(0, 0, "B"),
            sex = c(1, 2, 1)
        ), file = "test.fam", sep = "\t", quote = FALSE,
        row.names = FALSE, col.names = FALSE
    )
    fam <- "test.fam"
    pedi <- plink_to_pedigree(fam)
}
```

---

plot,Pedigree,missing-method

Plot Pedigrees

Description

This function is used to plot a Pedigree object.

It is a wrapper for `plot_fromdf()` and `ped_to_plotdf()` as well as `ped_to_legdf()` if `legend = TRUE`.

Usage

```
## S4 method for signature 'Pedigree,missing'
plot(
  x,
  aff_mark = TRUE,
  id_lab = "id",
  label = NULL,
  ggplot_gen = FALSE,
  cex = 1,
  symbolsize = 1,
  branch = 0.6,
  packed = TRUE,
  align = c(1.5, 2),
  align_parents = TRUE,
  force = FALSE,
  width = 6,
```

```
  title = NULL,
  subreg = NULL,
  pconnect = 0.5,
  fam_to_plot = 1,
  legend = FALSE,
  leg_cex = 0.8,
  leg_symbolsize = 0.5,
  leg_loc = NULL,
  leg_adjx = 0,
  leg_adjy = 0,
  precision = 4,
  lwd = 1,
  ped_par = list(),
  leg_par = list(),
  tips = NULL,
  title_cex = 2,
  leg_usr = NULL,
  add_to_existing = FALSE,
  label_dist = c(1, 3, 5),
  label_cex = c(1, 0.7, 1)
)
```

Arguments

x	A Pedigree object.
aff_mark	If TRUE, add a aff_mark to each box corresponding to the value of the affection column for each filling scale.
id_lab	The column name of the id for each individuals.
label	If not NULL, add a label to each box under the id corresponding to the value of the column given.
ggplot_gen	If TRUE, the function will use the ggplot2 package to generate the plot.
cex	Character expansion of the text
symbolsize	Size of the symbols
branch	defines how much angle is used to connect various levels of nuclear families.
packed	Should the Pedigree be compressed. (i.e. allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.)
align	For a packed Pedigree, align children under parents TRUE, to the extent possible given the page width, or align to to the left margin FALSE. This argument can be a two element vector, giving the alignment parameters, or a logical value. If TRUE, the default is c(1.5, 2), or if numeric the routine alignped4() will be called.
align_parents	If align_parents = TRUE, go one step further and try to make both parents of each child have the same depth. (This is not always possible). It helps the drawing program by lining up pedigrees that 'join in the middle' via a marriage.
force	If force = TRUE, the function will return the depth minus min(depth) if depth reach a state with no founders is not possible.

width	For a packed output, the minimum width of the plot, in inches.
title	The title of the plot.
subreg	A 4-element vector for (min x, max x, min depth, max depth), used to edit away portions of the plot coordinates returned by ped_to_plotdf(). This is useful for zooming in on a particular region of the Pedigree.
pconnect	When connecting parent to children the program will try to make the connecting line as close to vertical as possible, subject to it lying inside the endpoints of the line that connects the children by at least pconnect people. Setting this option to a large number will force the line to connect at the midpoint of the children.
fam_to_plot	default=1. If the Pedigree contains multiple families, this parameter can be used to select which family to plot. It can be a numeric value or a character value. If numeric, it is the index of the family to plot returned by unique(xpedfamid). If character, it is the family id to plot.
legend	default=FALSE. If TRUE, a legend will be added to the plot.
leg_cex	default=0.8. Controls the size of the legend text.
leg_symbolsize	default=0.5. Controls the size of the legend symbols.
leg_loc	default=NULL. If NULL, the legend will be placed in the upper right corner of the plot. Otherwise, a 4-element vector of the form (x0, x1, y0, y1) can be used to specify the location of the legend. The legend will be fitted to the specified and might be distorted if the aspect ratio of the legend is different from the aspect ratio of the specified location.
leg_adjx	default=0. Controls the horizontal labels adjustment of the legend.
leg_adjy	default=0. Controls the vertical labels adjustment of the legend.
precision	The number of significatif numbers to round the solution to.
lwd	default=1. Controls the line width of the segments, arcs and polygons.
ped_par	default=list(). A list of parameters to use as graphical parameters for the main plot.
leg_par	default=list(). A list of parameters to use as graphical parameters for the legend.
tips	A character vector of the column names of the data frame to use as tooltips. If NULL, no tooltips are added.
title_cex	The size of the title.
leg_usr	default=NULL. A vector of user coordinates to use for the legend.
add_to_existing	If TRUE, the plot will be added to the current plot.
label_dist	A numeric vector of length 3 giving the distance between the id, date and label text and the bottom of the box. This value is multiplied by the obtained labh value.
label_cex	A numeric vector of length 3 giving the cex of the id, date and label text. This value is multiplied by the cex argument

Details

Two important parameters control the looks of the result. One is the user specified maximum width. The smallest possible width is the maximum number of subjects on a line, if the user's suggestion is too low it is increased to 1 + that amount (to give just a little wiggle room).

To make a Pedigree where all children are centered under parents simply make the width large enough, however, the symbols may get very small.

The second is `align`, a vector of 2 alignment parameters `a` and `b`. For each set of siblings at a set of locations `x` and with parents at `p=c(p1,p2)` the alignment penalty is

$$(1/k^a) \sum i = 1k[(x_i - (p1 + p2)/2)]^2$$

$$\sum (x - \bar{(}p))^2/(k^a)$$

Where k is the number of siblings in the set.

When `a = 1` moving a sibship with `k` sibs one unit to the left or right of optimal will incur the same cost as moving one with only 1 or two sibs out of place.

If `a = 0` then large sibships are harder to move than small ones, with the default value `a = 1.5` they are slightly easier to move than small ones. The rationale for the default is as long as the parents are somewhere between the first and last siblings the result looks fairly good, so we are more flexible with the spacing of a large family. By tethering all the sibs to a single spot they are kept close to each other. The alignment penalty for spouses is $b(x_1 - x_2)^2$, which tends to keep them together. The size of `b` controls the relative importance of sib-parent and spouse-spouse closeness.

Value

an invisible list containing

- df : the data.frame used to plot the Pedigree
- par_usr : the user coordinates used to plot the Pedigree
- ggplot : the ggplot object if ggplot_gen = TRUE

Side Effects

Creates plot on current plotting device.

See Also

`Pedigree()`

Examples

```
data(sampleped)
pedAll <- Pedigree(sampleped)
if (interactive()) { plot(pedAll) }
```

plot_fromdf *Create a plot from a data.frame*

Description

This function is used to create a plot from a data.frame.

If `ggplot_gen = TRUE`, the plot will be generated with ggplot2 and will be returned invisibly.

Usage

```
plot_fromdf(
  df,
  usr = NULL,
  title = NULL,
  ggplot_gen = FALSE,
  boxw = 1,
  boxh = 1,
  add_to_existing = FALSE,
  title_cex = 2
)
```

Arguments

`df`	A data.frame with the following columns: • `type`: The type of element to plot. Can be `text`, `segments`, `arc` or other polygons. For polygons, the name of the polygon must be in the form `poly_*_*` where poly is one of the type given by `polygons()`, the first `*` is the number of slice in the polygon and the second `*` is the position of the division of the polygon. • `x0`: The x coordinate of the center of the element. • `y0`: The y coordinate of the center of the element. • `x1`: The x coordinate of the end of the element. Only used for `segments` and `arc`. • `y1`: The y coordinate of the end of the element. Only used for `segments` and `arc`. • `fill`: The fill color of the element. • `border`: The border color of the element. • `density`: The density of the element. • `angle`: The angle of the element. • `label`: The label of the element. Only used for `text`. • `cex`: The size of the element. • `adjx`: The x adjustment of the element. Only used for `text`. • `adjy`: The y adjustment of the element. Only used for `text`.
`usr`	The user coordinates of the plot.

title	The title of the plot.
ggplot_gen	If TRUE add the segments to the ggplot object
boxw	Width of the polygons elements
boxh	Height of the polygons elements
add_to_existing	If TRUE, the plot will be added to the current plot.
title_cex	The size of the title.

Value

an invisible ggplot object and a plot on the current plotting device

Examples

```
data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == 1,])
lst <- ped_to_plotdf(ped1)
if (interactive()) {
    plot_fromdf(lst$df, lst$par_usr$usr,
        boxw = lst$par_usr$boxw, boxh = lst$par_usr$boxh
    )
}
```

Rel-class *Rel object*

Description

S4 class to represent the special relationships in a Pedigree.

Constructor ::

You either need to provide a vector of the same size for each slot or a data.frame with the corresponding columns.

Usage

```
## S4 method for signature 'data.frame'
Rel(obj)

## S4 method for signature 'character_OR_integer'
Rel(obj, id2, code, famid = NA_character_, group = NA_character_)
```

Arguments

obj	A character vector with the id of the first individuals of each pairs or a data.frame with all the informations in corresponding columns.
id2	A character vector with the id of the second individuals of each pairs
code	A character, factor or numeric vector corresponding to the relation code of the individuals: • MZ twin = Monozygotic twin • DZ twin = Dizygotic twin • UZ twin = twin of unknown zygosity • Spouse = Spouse The following values are recognized: • character() or factor() : "MZ twin", "DZ twin", "UZ twin", "Spouse" with of without space between the words. The case is not important. • numeric() : 1 = "MZ twin", 2 = "DZ twin", 3 = "UZ twin", 4 = "Spouse"
famid	A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
group	A numeric vector with the set number for twins.

Details

A Rel object is a list of special relationships between individuals in the pedigree. It is used to create a Pedigree object. The minimal needed informations are id1, id2 and code.

If a famid is provided, the individuals id will be aggregated to the famid character to ensure the uniqueness of the id.

Value

A Rel object.

Slots

id1 A character vector with the id of the first individual.

id2 A character vector with the id of the second individual.

code An ordered factor vector with the code of the special relationship. (i.e. MZ twin < DZ twin < UZ twin < Spouse).

famid A character vector with the famid of the individuals.

group A numeric vector with the set number for twins.

Accessors

For all the following accessors, the x parameters is a Rel object. Each getters return a vector of the same length as x with the values of the corresponding slot.

- code(x) : Relationships' code
- id1(x) : Relationships' first individuals' identifier

- `id2(x)` : Relationships' second individuals' identifier
- `famid(x)` : Relationships' individuals' family identifier
- `famid(x) <- value` : Set the relationships' individuals' family identifier
  - `value` : A character or integer vector of the same length as x with the family identifiers

Generics

- `summary(x)`: Compute the summary of a Rel object
- `show(x)`: Convert the Rel object to a data.frame and print it with its summary.
- `as.list(x)`: Convert a Rel object to a list
- `as.data.frame(x)`: Convert a Rel object to a data.frame
- `subset(x, i, keep = TRUE)`: Subset a Rel object based on the individuals identifiers given.
  - `i` : A vector of individuals identifiers to keep.
  - `keep` : A logical value indicating if the individuals should be kept or deleted.

See Also

`Pedigree()`

Examples

```
rel_df <- data.frame(
    id1 = c("1", "2", "3"),
    id2 = c("2", "3", "4"),
    code = c(1, 1, 4)
)
Rel(rel_df)

Rel(
    obj = c("1", "2", "3"),
    id2 = c("2", "3", "4"),
    code = c(1, 1, 4)
)
```

relped *Relped data*

Description

Small set of related individuals for testing purposes.

Usage

```
data("relped")
```

Format

The dataframe is composed of 4 columns:

- id1 : the first individual identifier,
- id2 : the second individual identifier,
- code : the relationship between the two individuals,
- famid : the family identifier. The relationship codes are:
- 1 for Monozygotic twin
- 2 for Dizygotic twin
- 3 for Twin of unknown zygosity
- 4 for Spouse relationship

Details

This is a small fictive data set of relation that accompanies the sampleped data set. The aim was to create a data set with a variety of relationships. There is 8 relations with 4 different types of relationships.

Examples

```
data("relped")
data("sampleped")
pedi <- Pedigree(sampleped, relped)
summary(pedi)
if (interactive()) { plot(pedi) }
```

`sampleped` *Sampleped data*

Description

Small sample pedigree data set for testing purposes.

Usage

```
data("sampleped")
```

Format

A data frame with 55 observations, one line per subject, on the following 7 variables.

- `famid` : Family identifier
- `id` : Subject identifier
- `dadid` : Identifier of the father, if the father is part of the data set; zero otherwise
- `momid` : Identifier of the mother, if the mother is part of the data set; zero otherwise
- `sex` : `1` for male or `2` for female
- `affection` : `1` or `0`
- `avail` : `1` or `0`
- `num` : Numerical test variable from 0 to 6 randomly distributed

Details

This is a small fictive pedigree data set, with 55 individuals in 2 families. The aim was to create a data set with a variety of pedigree structures.

Examples

```
data("sampleped")
pedi <- Pedigree(sampleped)
summary(pedi)
if (interactive()) { plot(pedi) }
```

`Scales-class` *Scales object*

Description

A Scales object is a list of two data.frame. The first one is used to represent the affection status of the individuals and therefore the filling of the individuals in the pedigree plot. The second one is used to represent the availability status of the individuals and therefore the border color of the individuals in the pedigree plot.

Constructor ::
You need to provide both **fill** and **border** in the dedicated parameters. However this is usually done using the `generate_colors()` function with a Pedigree object.

Usage

```
Scales(fill, border)

## S4 method for signature 'data.frame,data.frame'
Scales(fill, border)
```

Arguments

`fill`	A data.frame with the informations for the affection status. The columns needed are: • 'order': the order of the affection to be used • 'column_values': name of the column containing the raw values in the Ped object • 'column_mods': name of the column containing the mods of the transformed values in the Ped object • 'mods': all the different mods • 'labels': the corresponding labels of each mods • 'affected': a logical value indicating if the mod correspond to an affected individuals • 'fill': the color to use for this mods • 'density': the density of the shading • 'angle': the angle of the shading
`border`	A data.frame with the informations for the availability status. The columns needed are: • 'column_values': name of the column containing the raw values in the Ped object • 'column_mods': name of the column containing the mods of the transformed values in the Ped object • 'mods': all the different mods • 'labels': the corresponding labels of each mods • 'border': the color to use for this mods

Value

A Scales object.

Slots

`fill` A data.frame with the informations for the affection status. The columns needed are:

- 'order': the order of the affection to be used
- 'column_values': name of the column containing the raw values in the Ped object
- 'column_mods': name of the column containing the mods of the transformed values in the Ped object
- 'mods': all the different mods
- 'labels': the corresponding labels of each mods
- 'affected': a logical value indicating if the mod correspond to an affected individuals
- 'fill': the color to use for this mods
- 'density': the density of the shading
- 'angle': the angle of the shading

`border` A data.frame with the informations for the availability status. The columns needed are:

- 'column_values': name of the column containing the raw values in the Ped object
- 'column_mods': name of the column containing the mods of the transformed values in the Ped object
- 'mods': all the different mods
- 'labels': the corresponding labels of each mods
- 'border': the color to use for this mods

Accessors

- `fill(x)` : Get the fill data.frame
- `fill(x) <- value` : Set the fill data.frame
- `border(x)` : Get the border data.frame
- `border(x) <- value` : Set the border data.frame from the Scales object.

Generics

- `as.list(x)`: Convert a Scales object to a list

See Also

`Pedigree()`

`generate_colors()`

Examples

```
Scales(
    fill = data.frame(
        order = 1,
        column_values = "affected",
        column_mods = "affected_mods",
        mods = c(0, 1),
        labels = c("unaffected", "affected"),
        affected = c(FALSE, TRUE),
        fill = c("white", "red"),
        density = c(NA, 20),
        angle = c(NA, 45)
    ),
    border = data.frame(
        column_values = "avail",
        column_mods = "avail_mods",
        mods = c(0, 1),
        labels = c("not available", "available"),
        border = c("black", "blue")
    )
)
```

---

shrink *Shrink Pedigree object*

Description

Shrink Pedigree object to specified bit size with priority placed on trimming uninformative subjects. The algorithm is useful for getting a Pedigree condensed to a minimally informative size for algorithms or testing that are limited by size of the Pedigree.

If **avail** or **affected** are NULL, they are extracted with their corresponding accessors from the Ped object.

Usage

```
## S4 method for signature 'Pedigree'
shrink(obj, avail = NULL, affected = NULL, max_bits = 16)

## S4 method for signature 'Ped'
shrink(obj, avail = NULL, affected = NULL, max_bits = 16)
```

Arguments

obj	A Pedigree or Ped object.
avail	A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
affected	A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).
max_bits	Optional, the bit size for which to shrink the Pedigree

Details

Iteratively remove subjects from the Pedigree. The random removal of members was previously controlled by a seed argument, but we remove this, forcing users to control randomness outside the function. First remove uninformative subjects, i.e., unavailable (not genotyped) with no available descendants. Next, available terminal subjects with unknown phenotype if both parents available. Last, iteratively shrinks Pedigrees by preferentially removing individuals (chosen at random if there are multiple of the same status):

1. Subjects with unknown affected status
2. Subjects with unaffected affected status
3. Affected subjects.

Value

A list containing the following elements:

- pedObj: Pedigree object after trimming
- id_trim: Vector of ids trimmed from Pedigree
- id_lst: List of ids trimmed by category
- bit_size: Vector of bit sizes after each trimming step
- avail: Vector of availability status after trimming
- pedSizeOriginal: Number of subjects in original Pedigree
- pedSizeIntermed: Number of subjects after initial trimming
- pedSizeFinal: Number of subjects after final trimming

Author(s)

Original by Dan Schaid, updated by Jason Sinnwell and Louis Le Nezet

See Also

Pedigree(), bit_size()

Examples

```
data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == '1',])
shrink(ped1, max_bits = 12)
```

`unrelated` *Find Unrelated subjects*

Description

Determine set of maximum number of unrelated available subjects from a Pedigree.

Usage

```
## S4 method for signature 'Ped'
unrelated(obj, avail = NULL)

## S4 method for signature 'Pedigree'
unrelated(obj, avail = NULL)
```

Arguments

`obj`	A Pedigree or Ped object.
`avail`	A logical vector with the availability status of the individuals (i.e. `FALSE` = not available, `TRUE` = available, `NA` = unknown).

Details

Determine set of maximum number of unrelated available subjects from a Pedigree, given vectors id, father, and mother for a Pedigree structure, and status vector of `TRUE` / `FALSE` for whether each subject is available (e.g. has DNA).

This is a greedy algorithm that uses the kinship matrix, sequentially removing rows/cols that are non-zero for subjects that have the most number of zero kinship coefficients (greedy by choosing a row of kinship matrix that has the most number of zeros, and then remove any cols and their corresponding rows that are non-zero. To account for ties of the count of zeros for rows, a random choice is made. Hence, running this function multiple times can return different sets of unrelated subjects.

If **avail** is `NULL`, it is extracted with its corresponding accessor from the Ped object.

Value

A vector of the ids of subjects that are unrelated.

Author(s)

Dan Schaid and Shannon McDonnell updated by Jason Sinnwell

Examples

```
data(sampleped)
fam1 <- sampleped[sampleped$famid == 1, -16]
ped1 <- Pedigree(fam1)
unrelated(ped1)
## some possible vectors
## [1] '110' '113' '133' '109'
## [1] '113' '118' '141' '109'
## [1] '113' '118' '140' '109'
## [1] '110' '113' '116' '109'
## [1] '113' '133' '141' '109'
```

---

upd_famid *Update family prefix in individuals id*

Description

Update the family prefix in the individuals identifiers. Individuals identifiers are constructed as follow **famid_id**. Therefore to update their family prefix the ids are split by the first underscore and the first part is overwritten by **famid**.

Usage

```
## S4 method for signature 'character,ANY'
upd_famid(obj, famid, missid = NA_character_)

## S4 method for signature 'Ped,character_OR_integer'
upd_famid(obj, famid)

## S4 method for signature 'Ped,missing'
upd_famid(obj)

## S4 method for signature 'Rel,character_OR_integer'
upd_famid(obj, famid)

## S4 method for signature 'Rel,missing'
upd_famid(obj)

## S4 method for signature 'Pedigree,character_OR_integer'
upd_famid(obj, famid)

## S4 method for signature 'Pedigree,missing'
upd_famid(obj)
```

Arguments

obj	Ped or Pedigree object or a character vector of individual ids
famid	A character vector with the family identifiers of the individuals. If provide, will be aggregated to the individuals identifiers separated by an underscore.
missid	A character vector with the missing values identifiers. All the id, dadid and momid corresponding to those values will be set to NA_character_.

Details

If famid is *missing*, then the famid() function will be called on the object.

Value

A character vector of individual ids with family prefix updated

Examples

```
upd_famid(c("1", "2", "B_3"), c("A", "B", "A"))
upd_famid(c("1", "B_2", "C_3", "4"), c("A", NA, "A", NA))

data(sampleped)
ped1 <- Pedigree(sampleped[,-1])
id(ped(ped1))
new_fam <- make_famid(id(ped(ped1)), dadid(ped(ped1)), momid(ped(ped1)))
id(ped(upd_famid(ped1, new_fam)))

data(sampleped)
ped1 <- Pedigree(sampleped[,-1])
make_famid(ped1)
```

---

useful_inds *Usefulness of individuals*

Description

Compute the usefulness of individuals

Usage

```
## S4 method for signature 'character'
useful_inds(
  obj,
  dadid,
  momid,
  avail,
  affected,
  num_child_tot,
```

```
  id_inf,
  keep_infos = FALSE
)

## S4 method for signature 'Pedigree'
useful_inds(obj, keep_infos = FALSE, reset = FALSE, max_dist = NULL)

## S4 method for signature 'Ped'
useful_inds(obj, keep_infos = FALSE, reset = FALSE, max_dist = NULL)
```

Arguments

obj	A character vector with the id of the individuals or a data.frame with all the informations in corresponding columns.
dadid	A vector containing for each subject, the identifiers of the biologicals fathers.
momid	A vector containing for each subject, the identifiers of the biologicals mothers.
avail	A logical vector with the availability status of the individuals (i.e. FALSE = not available, TRUE = available, NA = unknown).
affected	A logical vector with the affection status of the individuals (i.e. FALSE = unaffected, TRUE = affected, NA = unknown).
num_child_tot	A numeric vector of the number of children of each individuals
id_inf	An identifiers vector of informative individuals.
keep_infos	Boolean to indicate if parents with unknown status but available or reverse should be kept
reset	Boolean to indicate if the useful column should be reset
max_dist	The maximum distance to informative individuals

Details

Check for the informativeness of the individuals based on the informative parameter given, the number of children and the usefulness of their parents. A useful slot is added to the Ped object with the usefulness of the individual.

Value

When obj is a vector:
A vector of useful individuals identifiers

When obj is a Pedigree or Ped object:
The Pedigree or Ped object with the slot 'useful' containing TRUE for useful individuals and FALSE otherwise.

Examples

```
data(sampleped)
ped1 <- Pedigree(sampleped[sampleped$famid == "1",])
ped1 <- is_informative(ped1, informative = "AvAf", col_aff = "affection")
ped(useful_inds(ped1))
```

Index

∗ **Pedigree-plot**
ped_to_legdf, 54
ped_to_plotdf, 55
plot,Pedigree,missing-method, 66
plot_fromdf, 70
∗ **alignment,**
auto_hint, 13
best_hint, 15
∗ **alignment**
alignped1, 7
alignped2, 9
alignped3, 10
alignped4, 12
∗ **auto_hint**
auto_hint, 13
best_hint, 15
∗ **datasets**
minnbreast, 38
relped, 74
sampleped, 75
∗ **generate_scales**
generate_colors, 23
∗ **internal,**
alignped1, 7
alignped2, 9
alignped3, 10
alignped4, 12
auto_hint, 13
find_avail_affected, 18
find_avail_noninform, 19
find_unavailable, 20
ped_to_legdf, 54
ped_to_plotdf, 55
plot_fromdf, 70
∗ **ped_avaf_infos**
family_infos_table, 18
∗ **shrink**
bit_size, 17
find_avail_affected, 18
find_avail_noninform, 19
find_unavailable, 20
shrink, 78
useful_inds, 82
[,Pedigree,ANY,missing,ANY-method *(*Pedigree-class*)*, 58

adopted *(*Ped-class*)*, 46
adopted,Ped-method *(*Ped-class*)*, 46
adopted<- *(*Ped-class*)*, 46
adopted<-,Ped,numeric_OR_logical-method *(*Ped-class*)*, 46
affected *(*Ped-class*)*, 46
affected,Ped-method *(*Ped-class*)*, 46
affected<- *(*Ped-class*)*, 46
affected<-,Ped,numeric_OR_logical-method *(*Ped-class*)*, 46
align, 5
align(), *8*, *10*, *11*, *13*, *15*, *16*, *33*
align,Pedigree-method *(*align*)*, 5
alignped1, 7
alignped1(), *6*
alignped2, 9
alignped2(), *6*, *8*
alignped3, 10
alignped3(), *6*
alignped4, 12
alignped4(), *6*
as.data.frame,Ped-method *(*Ped-class*)*, 46
as.data.frame,Rel-method *(*Rel-class*)*, 71
as.list,Hints-method *(*Hints-class*)*, 26
as.list,Ped-method *(*Ped-class*)*, 46
as.list,Pedigree-method *(*Pedigree-class*)*, 58
as.list,Rel-method *(*Rel-class*)*, 71
as.list,Scales-method *(*Scales-class*)*, 76
asymptomatic *(*Ped-class*)*, 46
asymptomatic,Ped-method *(*Ped-class*)*, 46
asymptomatic<- *(*Ped-class*)*, 46

asymptomatic<-,Ped,numeric_OR_logical-method (Ped-class), 46
auto_hint, 13
auto_hint(), *6*, *16*
auto_hint,Pedigree-method (auto_hint), 13
avail (Ped-class), 46
avail,Ped-method (Ped-class), 46
avail<- (Ped-class), 46
avail<-,Ped,numeric_OR_logical-method (Ped-class), 46

best_hint, 15
best_hint(), *15*
best_hint,Pedigree-method (best_hint), 15
bit_size, 17
bit_size(), *4*, *79*
bit_size,character_OR_integer-method (bit_size), 17
bit_size,Ped-method (bit_size), 17
bit_size,Pedigree-method (bit_size), 17
border (Scales-class), 76
border,Pedigree-method (Pedigree-class), 58
border,Scales-method (Scales-class), 76
border<- (Scales-class), 76
border<-,Pedigree,data.frame-method (Pedigree-class), 58
border<-,Scales,data.frame-method (Scales-class), 76

carrier (Ped-class), 46
carrier,Ped-method (Ped-class), 46
carrier<- (Ped-class), 46
carrier<-,Ped,numeric_OR_logical-method (Ped-class), 46
char_to_date(), *40*, *63*
code (Rel-class), 71
code,Rel-method (Rel-class), 71
complete_twins(), *43*
consultand (Ped-class), 46
consultand,Ped-method (Ped-class), 46
consultand<- (Ped-class), 46
consultand<-,Ped,numeric_OR_logical-method (Ped-class), 46

dadid (Ped-class), 46
dadid,Ped-method (Ped-class), 46
dadid<- (Ped-class), 46
dadid<-,Ped,character_OR_integer-method (Ped-class), 46
dateofbirth (Ped-class), 46
dateofbirth,Ped-method (Ped-class), 46
dateofbirth<- (Ped-class), 46
dateofbirth<-,Ped,Date_OR_character-method (Ped-class), 46
dateofdeath (Ped-class), 46
dateofdeath,Ped-method (Ped-class), 46
dateofdeath<- (Ped-class), 46
dateofdeath<-,Ped,Date_OR_character-method (Ped-class), 46
deceased (Ped-class), 46
deceased,Ped-method (Ped-class), 46
deceased<- (Ped-class), 46
deceased<-,Ped,numeric_OR_logical-method (Ped-class), 46

evaluated (Ped-class), 46
evaluated,Ped-method (Ped-class), 46
evaluated<- (Ped-class), 46
evaluated<-,Ped,numeric_OR_logical-method (Ped-class), 46
exclude_stray_marryin(), *21*
exclude_unavail_founders(), *21*

famid (Ped-class), 46
famid,Ped-method (Ped-class), 46
famid,Rel-method (Rel-class), 71
famid<- (Ped-class), 46
famid<-,Ped,character_OR_integer-method (Ped-class), 46
famid<-,Rel,character_OR_integer-method (Rel-class), 71
family_infos_table, 18
fertility (Ped-class), 46
fertility,Ped-method (Ped-class), 46
fertility<- (Ped-class), 46
fertility<-,Ped,character_OR_integer-method (Ped-class), 46
fertility_to_factor(), *40*, *63*
fill (Scales-class), 76
fill,Pedigree-method (Pedigree-class), 58
fill,Scales-method (Scales-class), 76
fill<- (Scales-class), 76
fill<-,Pedigree,data.frame-method (Pedigree-class), 58

fill<-,Scales,data.frame-method (Scales-class), 76
find_avail_affected, 18
find_avail_affected,Ped-method (find_avail_affected), 18
find_avail_affected,Pedigree-method (find_avail_affected), 18
find_avail_noninform, 19
find_avail_noninform,Ped-method (find_avail_noninform), 19
find_avail_noninform,Pedigree-method (find_avail_noninform), 19
find_unavailable, 20
find_unavailable,Ped-method (find_unavailable), 20
find_unavailable,Pedigree-method (find_unavailable), 20
fix_parents, 22
fix_parents,character-method (fix_parents), 22
fix_parents,data.frame-method (fix_parents), 22

generate_border(), *25*
generate_colors, 23, *60*
generate_colors(), *76, 77*
generate_colors,character-method (generate_colors), 23
generate_colors,numeric-method (generate_colors), 23
generate_colors,Pedigree-method (generate_colors), 23

Hints, *14, 15*
Hints (Hints-class), 26
hints (Pedigree-class), 58
Hints(), *63, 64*
Hints,Hints,missing_OR_NULL-method (Hints-class), 26
Hints,list,missing_OR_NULL-method (Hints-class), 26
Hints,missing_OR_NULL,data.frame-method (Hints-class), 26
Hints,missing_OR_NULL,missing_OR_NULL-method (Hints-class), 26
Hints,numeric,data.frame-method (Hints-class), 26
Hints,numeric,missing_OR_NULL-method (Hints-class), 26
hints,Pedigree-method (Pedigree-class), 58
Hints-class, 26
hints<- (Pedigree-class), 58
hints<-,Pedigree,Hints-method (Pedigree-class), 58
horder (Hints-class), 26
horder,Hints-method (Hints-class), 26
horder,Pedigree-method (Pedigree-class), 58
horder<- (Hints-class), 26
horder<-,Hints-method (Hints-class), 26
horder<-,Pedigree-method (Pedigree-class), 58

ibd_matrix, 28
id (Ped-class), 46
id,Ped-method (Ped-class), 46
id1 (Rel-class), 71
id1,Rel-method (Rel-class), 71
id2 (Rel-class), 71
id2,Rel-method (Rel-class), 71
id<- (Ped-class), 46
id<-,Ped,character_OR_integer-method (Ped-class), 46
is_informative, 29
is_informative,character_OR_integer-method (is_informative), 29
is_informative,Ped-method (is_informative), 29
is_informative,Pedigree-method (is_informative), 29
is_parent, 31
is_parent,character_OR_integer-method (is_parent), 31
is_parent,Ped-method (is_parent), 31
isinf (Ped-class), 46
isinf,Ped-method (Ped-class), 46
isinf<- (Ped-class), 46
isinf<-,Ped,numeric_OR_logical-method (Ped-class), 46

kin (Ped-class), 46
kin,Ped-method (Ped-class), 46
kin<- (Ped-class), 46
kin<-,Ped,numeric-method (Ped-class), 46
kindepth, 32
kindepth(), *34*

kindepth,character_OR_integer-method (kindepth), 32
kindepth,Ped-method (kindepth), 32
kindepth,Pedigree-method (kindepth), 32
kinship, 33
kinship(), *4*, *29*, *36, 37*
kinship,character-method (kinship), 33
kinship,Ped-method (kinship), 33
kinship,Pedigree-method (kinship), 33

length,Pedigree-method (Pedigree-class), 58

make_famid, 35
make_famid(), *34*
make_famid,character-method (make_famid), 35
make_famid,Pedigree-method (make_famid), 35
mcols,Pedigree-method (Pedigree-class), 58
mcols<-,Ped,data.frame-method (Ped-class), 46
mcols<-,Ped,list-method (Ped-class), 46
mcols<-,Pedigree,ANY-method (Pedigree-class), 58
min_dist_inf, 36
min_dist_inf,character-method (min_dist_inf), 36
min_dist_inf,Ped-method (min_dist_inf), 36
min_dist_inf,Pedigree-method (min_dist_inf), 36
minnbreast, 38
minnbreast(), *4*
miscarriage (Ped-class), 46
miscarriage,Ped-method (Ped-class), 46
miscarriage<- (Ped-class), 46
miscarriage<-,Ped,character_OR_integer-method (Ped-class), 46
miscarriage_to_factor(), *40*, *63*
momid (Ped-class), 46
momid,Ped-method (Ped-class), 46
momid<- (Ped-class), 46
momid<-,Ped,character_OR_integer-method (Ped-class), 46

norm_ped, 40
norm_rel, 42
num_child, 44
num_child,character_OR_integer-method (num_child), 44
num_child,Pedigree-method (num_child), 44

par(), *56*
parent_of, 45
parent_of,character_OR_integer-method (parent_of), 45
parent_of,Ped-method (parent_of), 45
parent_of,Pedigree-method (parent_of), 45
Ped, *41*
Ped (Ped-class), 46
ped (Pedigree-class), 58
Ped(), *41*, *60*, *63, 64*
Ped,character_OR_integer-method (Ped-class), 46
Ped,data.frame-method (Ped-class), 46
Ped,missing-method (Ped-class), 46
ped,Pedigree,ANY-method (Pedigree-class), 58
ped,Pedigree,missing-method (Pedigree-class), 58
Ped-class, 46
ped<- (Pedigree-class), 58
ped<-,Pedigree,ANY,ANY-method (Pedigree-class), 58
ped<-,Pedigree,missing,Ped-method (Pedigree-class), 58
ped_server, 52
ped_shiny, 53
ped_to_legdf, 54
ped_to_legdf(), *4*, *57*, *66*
ped_to_legdf,Pedigree-method (ped_to_legdf), 54
ped_to_plotdf, 55
ped_to_plotdf(), *6*, *66*, *68*
ped_to_plotdf,Pedigree-method (ped_to_plotdf), 55
ped_ui, 58
Pedigree (Pedigree-class), 58
Pedigree(), *4*, *28*, *41*, *51*, *64*, *69*, *73*, *77*, *79*
Pedigree,character_OR_integer-method (Pedigree-class), 58
Pedigree,data.frame-method (Pedigree-class), 58

Pedigree,missing-method
(Pedigree-class), 58
Pedigree-class, 58
Pedixplorer (Pedixplorer-package), 3
Pedixplorer-package, 3
plink_to_pedigree, 65
plot(), *4*
plot,Pedigree
(plot,Pedigree,missing-method),
66
plot,Pedigree,missing-method, 66
plot.Pedigree
(plot,Pedigree,missing-method),
66
plot_fromdf, 70
plot_fromdf(), *4*, *54, 55*, *57*, *66*
polygons(), *70*
proband (Ped-class), 46
proband,Ped-method (Ped-class), 46
proband<- (Ped-class), 46
proband<-,Ped,numeric_OR_logical-method
(Ped-class), 46

Rel (Rel-class), 71
rel (Pedigree-class), 58
Rel(), *43, 44*, *61*, *63, 64*
Rel,character_OR_integer-method
(Rel-class), 71
Rel,data.frame-method (Rel-class), 71
Rel,missing-method (Rel-class), 71
rel,Pedigree,ANY-method
(Pedigree-class), 58
rel,Pedigree,missing-method
(Pedigree-class), 58
Rel-class, 71
rel<- (Pedigree-class), 58
rel<-,Pedigree,ANY,ANY-method
(Pedigree-class), 58
rel<-,Pedigree,missing,Rel-method
(Pedigree-class), 58
rel_code_to_factor(), *43, 44*, *61*
relped, 74

sampleped, 75
sampleped(), *4*
Scales (Scales-class), 76
scales (Pedigree-class), 58
Scales(), *63, 64*
Scales,data.frame,data.frame-method
(Scales-class), 76
Scales,missing,missing-method
(Scales-class), 76
scales,Pedigree-method
(Pedigree-class), 58
Scales-class, 76
scales<- (Pedigree-class), 58
scales<-,Pedigree,Scales-method
(Pedigree-class), 58
sex (Ped-class), 46
sex,Ped-method (Ped-class), 46
sex<- (Ped-class), 46
sex<-,Ped,character_OR_integer-method
(Ped-class), 46
show,Ped-method (Ped-class), 46
show,Pedigree-method (Pedigree-class),
58
show,Rel-method (Rel-class), 71
shrink, 78
shrink(), *4*, *17*, *19–21*
shrink,Ped-method (shrink), 78
shrink,Pedigree-method (shrink), 78
spouse (Hints-class), 26
spouse,Hints-method (Hints-class), 26
spouse,Pedigree-method
(Pedigree-class), 58
spouse<- (Pedigree-class), 58
spouse<-,Hints,data.frame-method
(Hints-class), 26
spouse<-,Pedigree,data.frame-method
(Pedigree-class), 58
subset,Hints-method (Hints-class), 26
subset,Ped-method (Ped-class), 46
subset,Pedigree-method
(Pedigree-class), 58
subset,Rel-method (Rel-class), 71
summary,Ped-method (Ped-class), 46
summary,Pedigree-method
(Pedigree-class), 58
summary,Rel-method (Rel-class), 71

unrelated, 80
unrelated,Ped-method (unrelated), 80
unrelated,Pedigree-method (unrelated),
80
upd_famid, 81
upd_famid(), *40*, *43*

upd_famid,character,ANY-method
(upd_famid), 81
upd_famid,Ped,character_OR_integer-method
(upd_famid), 81
upd_famid,Ped,missing-method
(upd_famid), 81
upd_famid,Pedigree,character_OR_integer-method
(upd_famid), 81
upd_famid,Pedigree,missing-method
(upd_famid), 81
upd_famid,Rel,character_OR_integer-method
(upd_famid), 81
upd_famid,Rel,missing-method
(upd_famid), 81
useful (Ped-class), 46
useful,Ped-method (Ped-class), 46
useful<- (Ped-class), 46
useful<-,Ped,numeric_OR_logical-method
(Ped-class), 46
useful_inds, 82
useful_inds,character-method
(useful_inds), 82
useful_inds,Ped-method (useful_inds), 82
useful_inds,Pedigree-method
(useful_inds), 82

vect_to_binary(), *40*, *63*