Pedigree handling

Gregor Gorjanc
gregor.gorjanc@bfro.uni-lj.si,
David A. Henderson
dnadave@insightful.com

April 27, 2025

Introduction

Pedigrees are collections of related individuals. Often we represent these as a linked list, a collection of trios that links (or almost everyone) everyone together: an individual and its two parents. This simple representation allows the use of graph theory in analysis. The GeneticsPed package provides utilities for managing pedigrees; inputing, sorting, and subsetting pedigrees; and computing on pedigrees by calculating relationship coefficients and other similar quantities.

name some fields were pedigree is used Falconer and Mackay (1996)

?

Pedigree class

describe

individual subject ascendant

can be factor, character, numeric, but all must have the same class

Unknown individuals

FIXME Pedigrees are never complete because it is not possible to get data on all ascendants. Therefore there are always some subjects with unknown ascendants. As with pedigree form there are also differences in representation of unknown individuals between different applications, namely using 0, blank field, particular string as "unknown", etc. In GeneticsPed R's unknown representation `NA` is used. Change from other representations to `NA` can be done prior to definition of a pedigree object. To ease this process, we have provided argument `unknown` in `Pedigree`. Multiple values can be passed to that argument in one

call i.e. `uknown=c(0, "", "unknown")`. Internally, change is done via `uknownToNA` generic function. In case one wants to use some other representation for example for special application in R or exporting to outer application, `NAToUknown` function is provided.

How will we handle if one wants anything else than 0 in R - should we allow for this or just convert each time to NA in our functions?

```
> 1+1
```

```
[1] 2
```

1 Check consistency of data in pedigree

check

check.Pedigree checkId

check performs a series of checks on pedigree object to ensure consistency of data.

check(x, ...) checkId(x)

x pedigree, object to be checked

...] arguments to other methods, none for now

checkId performs various checks on subjects and their ascendants. These checks are:

- idClass: all ids must have the same class
- subjectIsNA: subject can not be NA
- subjectNotUnique: subject must be unique
- subjectEqualAscendant: subject can not be equal (in identification) to its ascendant
- ascendantEqualAscendant: ascendant can not be equal to another ascendant
- ascendantInAscendant: ascendant can not appear again as asescendant of other sex i.e. father can not be a mother to someone else
- unusedLevels: in case factors are used for id presentation, there might be unused levels for some ids - some functions rely on number of levels and a check is provided for this

checkAttributes is intended primarly for internal use and performs a series of checks on attribute values needed in various functions. It causes stop with error messages for all given attribute checks.

List of more or less self-explanatory errors and "pointers" to these errors for ease of further work i.e. removing errors.

```
## EXAMPLES BELLOW ARE ONLY FOR TESTING PURPOSES AND ARE NOT INTENDED
## FOR USERS, BUT IT CAN NOT DO ANY HARM.

## --- checkAttributes ---
tmp <- generatePedigree(5)
attr(tmp, "sorted") <- FALSE
attr(tmp, "coded") <- FALSE
GeneticsPed:::checkAttributes(tmp)
try(GeneticsPed:::checkAttributes(tmp, sorted=TRUE, coded=TRUE))

## --- idClass ---
tmp <- generatePedigree(5)
tmp$id <- factor(tmp$id)
class(tmp$id)
class(tmp$father)
try(GeneticsPed:::idClass(tmp))

## --- subjectIsNA ---
tmp <- generatePedigree(2)
tmp[1, 1] <- NA
GeneticsPed:::subjectIsNA(tmp)

## --- subjectNotUnique ---
tmp <- generatePedigree(2)
tmp[2, 1] <- 1
GeneticsPed:::subjectNotUnique(tmp)

## --- subjectEqualAscendant ---
tmp <- generatePedigree(2)
tmp[3, 2] <- tmp[3, 1]
GeneticsPed:::subjectEqualAscendant(tmp)

## --- ascendantEqualAscendant ---
tmp <- generatePedigree(2)
tmp[3, 2] <- tmp[3, 3]
GeneticsPed:::ascendantEqualAscendant(tmp)

## --- ascendantInAscendant ---
tmp <- generatePedigree(2)
tmp[3, 2] <- tmp[5, 3]
GeneticsPed:::ascendantInAscendant(tmp)
## Example with multiple parents
tmp <- data.frame(id=c("A", "B", "C", "D"),
                  father1=c("E", NA, "F", "H"),
                  father2=c("F", "E", "E", "I"),
```

```
                  mother=c("G", NA, "H", "E"))
tmp <- Pedigree(tmp, ascendant=c("father1", "father2", "mother"),
                ascendantSex=c(1, 1, 2),
                ascendantLevel=c(1, 1, 1))
GeneticsPed:::ascendantInAscendant(tmp)

## --- unusedLevels ---
tmp <- generatePedigree(2, colClass="factor")
tmp[3:4, 2] <- NA
GeneticsPed:::unusedLevels(tmp)
```

References

Falconer, D. S. and Mackay, T. F. C. (1996). *Introduction to Quantitative Genetics*. Longman, Essex, U.K., 4th ed. edition.