Package: BreedingExperiment (via r-universe)

November 24, 2026

Title Container and Relationship Matrices for Breeding and Quantitative Genomics

Version 0.99.5

Description Provides the BreedingExperiment class, a container that keeps marker genotypes, phenotypes, pedigree and marker coordinates together in one object, extending RangedSummarizedExperiment so that markers are addressable as genomic ranges. Relationship matrices central to quantitative genetics are computed as methods on the container: the pedigree numerator relationship matrix of Henderson (1976) <doi:10.2307/2529339>, the genomic relationship matrix of VanRaden (2008) <doi:10.3168/jds.2007-0980>, the dominance relationship matrix of Vitezica, Varona and Legarra (2013) <doi:10.1534/genetics.113.155176>, and the single-step matrix combining pedigree and genomic information of Legarra, Aguilar and Misztal (2009) <doi:10.3168/jds.2009-2061> and Christensen and Lund (2010) <doi:10.1186/1297-9686-42-2>, which allows genotyped and ungenotyped individuals to be analysed together as in Aguilar and others (2010) <doi:10.3168/jds.2009-2730>. Marker quality control, imputation, inbreeding coefficients and allele frequency summaries are included.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.5.0), SummarizedExperiment

Imports methods, stats, utils, S4Vectors, GenomicRanges, IRanges, BiocGenerics

Suggests testthat (>= 3.0.0), knitr, rmarkdown, BiocStyle

VignetteBuilder knitr

biocViews Software, Genetics, SNP, GenomicVariation, DataRepresentation, Infrastructure, Regression, QualityControl

URL https://github.com/mqfarooqi1/BreedingExperiment, https://mqfarooqi1.github.io/BreedingExperiment/

BugReports https://github.com/mqfarooqi1/BreedingExperiment/issues

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Collate 'AllClasses.R' 'AllGenerics.R' 'BreedingExperiment-package.R' 'constructor.R' 'accessors.R' 'pedigree.R' 'relationship.R' 'singlestep.R' 'qc.R' 'simulate.R' 'data.R' 'show.R'

Config/roxygen2/version 8.1.0

Config/pak/sysreqs zlib1g-dev

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

Date/Publication 2026-11-23 06:34:54 UTC

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

RemoteRef HEAD

RemoteSha a98d16c3b9298eba61c30035949c27754fedf597

Contents

alleleFrequency . . . 2
Amatrix . . . 4
BreedingExperiment . . . 5
BreedingExperiment-class . . . 6
demoBreeding . . . 8
Dmatrix . . . 9
filterMarkers . . . 10
genotyped . . . 11
genotypes . . . 12
Gmatrix . . . 13
Hmatrix . . . 14
imputeMarkers . . . 16
inbreeding . . . 17
pedigree . . . 18
simulateBreeding . . . 18
sortPedigree . . . 20

Index **21**

`alleleFrequency` *Allele frequencies and marker summaries*

Description

Computes, for every marker, the frequency of the counted allele, the minor allele frequency, the call rate and the observed heterozygosity.

Usage

```
alleleFrequency(x, ...)

## S4 method for signature 'BreedingExperiment'
alleleFrequency(x, tol = 1e-08, ...)
```

Arguments

x	A BreedingExperiment object.
...	Unused.
tol	How far a dosage may sit from a whole number and still count as a genotype call when heterozygosity is computed.

Details

freq and maf use every dosage that is present, so an expected dosage from imputation contributes to them on the same footing as a call. callRate is the proportion of dosages that are not NA; after imputeMarkers() that is 1 for every marker, so take call rates before imputing if you need them.

heterozygosity is different, because a heterozygote is a genotype rather than a quantity. A cell counts towards it only if it is a genotype call: its dosage must be a whole number, and it must not be marked in the imputed assay that imputeMarkers() writes. Everything else is left out of both the numerator and the denominator. Without those two checks an imputed cell would be counted as a non-heterozygote and heterozygosity would appear to fall every time a data set was imputed. A marker with no genotype call at all gets NA.

Dosages imputed by other software arrive without the imputed assay, so only the whole-number check applies to them; pass such dosages in as fractional values, or carry your own mask as an extra assay, if observed heterozygosity matters to you.

Value

A S4Vectors::DataFrame with one row per marker and columns freq, maf, callRate and heterozygosity.

See Also

filterMarkers(), imputeMarkers()

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 30, n_marker = 50)
head(alleleFrequency(be))

# imputation does not change the observed heterozygosity of a marker,
# because imputed cells are not counted as genotype calls
bem <- simulateBreeding(n_ind = 30, n_marker = 50, missing = 0.2)
all.equal(alleleFrequency(bem)$heterozygosity,
          alleleFrequency(imputeMarkers(bem))$heterozygosity)
```

`Amatrix` *Pedigree numerator relationship matrix*

Description

Builds the additive relationship matrix A from the pedigree using the tabular method. Element A_{ij} is twice the coefficient of kinship between individuals i and j, so a diagonal entry is $1 + F_i$ where F_i is the inbreeding coefficient.

Usage

```
Amatrix(x, ...)

## S4 method for signature 'BreedingExperiment'
Amatrix(x, ids = NULL, ...)
```

Arguments

`x`	A BreedingExperiment object.
`...`	Unused.
`ids`	Optional character vector of identifiers to return, in that order. Defaults to every individual in the pedigree.

Details

The matrix is built recursively over a pedigree sorted so that parents come first: an individual's relationship to an earlier individual is the average of its parents' relationships to that individual, and its diagonal is $1 + \frac{1}{2} A_{sire,dam}$.

Value

A symmetric numeric matrix with dimnames taken from the pedigree.

References

Henderson, C. R. (1976) "A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values." Biometrics 32, 69-83. doi:10.2307/2529339

See Also

`Gmatrix()`, `Hmatrix()`, `inbreeding()`

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 12, n_marker = 30)
A <- Amatrix(be)
round(A[1:5, 1:5], 3)
```

`BreedingExperiment` *Create a breeding experiment*

Description

Builds a BreedingExperiment from a marker genotype matrix, with optional phenotypes, pedigree and marker coordinates.

Usage

```
BreedingExperiment(
  genotypes,
  phenotypes = NULL,
  pedigree = NULL,
  rowRanges = NULL,
  ...
)
```

Arguments

`genotypes`	A numeric matrix of allele dosages, markers in rows and individuals in columns, with column names identifying the individuals.
`phenotypes`	Optional data frame or S4Vectors::DataFrame of individual-level variables, with one row per column of `genotypes`.
`pedigree`	Optional data frame with columns `id`, `sire` and `dam`. Unknown parents may be `NA` or `0`.
`rowRanges`	Optional GenomicRanges::GRanges giving the position of each marker, of the same length as `nrow(genotypes)`.
`...`	Further assays, passed to `SummarizedExperiment::SummarizedExperiment()`.

Details

Genotypes are allele dosages: usually the integers 0, 1 and 2, with `NA` for missing calls. Fractional values in between are accepted, so expected dosages from imputation can be stored directly. Markers are rows and individuals are columns, matching the Bioconductor convention that features are rows.

If `rowRanges` is not supplied, markers are given placeholder coordinates on a single sequence so that the object is still a valid ranged experiment; pass real coordinates whenever you have them, because that is what allows a data set to be subset by genomic region.

The pedigree may name ancestors that were never genotyped. Those extra individuals are exactly what `Hmatrix()` needs in order to combine pedigree and genomic information.

Value

A BreedingExperiment object.

How phenotypes are matched to individuals

If `phenotypes` has row names, they are treated as individual identifiers: they must be the same set as the column names of `genotypes`, and the rows are reordered to match. If it has no row names, or only the automatic `1, 2, 3, ...` of a freshly built data frame, the rows are taken in the order given. Row names that are neither absent nor matching are an error rather than something to be overwritten, because the one thing worse than refusing a phenotype table is silently attaching each phenotype to the wrong individual.

See Also

`Gmatrix()`, `Amatrix()`, `Hmatrix()`, `simulateBreeding()`

Examples

```
set.seed(1)
geno <- matrix(rbinom(40 * 12, 2, 0.3), nrow = 40, ncol = 12,
               dimnames = list(paste0("snp", 1:40), paste0("ind", 1:12)))
pheno <- data.frame(yield = rnorm(12))
be <- BreedingExperiment(geno, phenotypes = pheno)
be
dim(be)

# phenotypes given in a different order are matched by row name
shuffled <- data.frame(yield = rnorm(12),
                       row.names = rev(colnames(geno)))
identical(colnames(BreedingExperiment(geno, phenotypes = shuffled)),
          colnames(geno))
```

`BreedingExperiment-class` *Breeding experiment container*

Description

A `BreedingExperiment` keeps everything a quantitative-genetic analysis needs in one object: marker genotypes, the phenotypes and design variables of the individuals, the pedigree, and the position of every marker on the genome.

Usage

```
## S4 method for signature 'BreedingExperiment'
show(object)
```

Arguments

`object`	A BreedingExperiment object.

Details

The class extends SummarizedExperiment::RangedSummarizedExperiment, so markers are rows and individuals are columns, and the familiar accessors work as usual: `assay()` returns the genotype matrix, `rowRanges()` the marker coordinates as a GenomicRanges::GRanges, and `colData()` the phenotypes. Because marker positions are genomic ranges, a breeding data set can be subset by region, overlapped with annotation, and otherwise handled with the standard Bioconductor vocabulary.

One slot is added to the parent class:

`pedigree` A S4Vectors::DataFrame with columns `id`, `sire` and `dam`, giving the parents of each individual. Unknown parents are `NA`. It may contain ancestors that were never genotyped, which is what makes single-step analysis possible.

Genotypes are stored as allele dosages: 0, 1 or 2 copies of the counted allele, with `NA` for missing calls.

Value

A class definition, so nothing is returned by the class itself. The constructor `BreedingExperiment()` returns a `BreedingExperiment` object, and the `show` method is called for the summary it prints, returning `object` invisibly.

Construction

Use `BreedingExperiment()`; see its help page for examples.

Display

The `show` method prints the size of the object, the assays it holds, the phenotype columns, the sequences the markers lie on, and a summary of the pedigree including how many individuals are genotyped.

See Also

`BreedingExperiment()`, `Amatrix()`, `Gmatrix()`, `Hmatrix()`

Examples

```
data(demoBreeding)
demoBreeding
```

`demoBreeding` *A demonstration breeding population*

Description

A small, five-generation breeding population supplied so that every function in the package can be tried on data that behaves like a real programme.

Usage

```
data(demoBreeding)
```

Format

A BreedingExperiment with 900 markers and 120 genotyped individuals, and a pedigree of 180. `colData` holds:

generation Generation number, 3 to 5 for genotyped individuals.

sex `"F"` or `"M"`.

yield A moderately heritable trait ($h^2 \approx 0.35$), recorded on females only.

stature A more heritable trait ($h^2 \approx 0.60$), recorded on everyone.

trueBV_yield, trueBV_stature The simulated true breeding values.

Details

The data are **simulated**, not observed. They are built to have the awkward features that real breeding data has and that tidy textbook examples lack:

- **Only part of the population is genotyped.** The two earliest generations were born before genotyping began, so 60 of the 180 individuals in the pedigree have no markers. This is the situation `Hmatrix()` exists for.
- **Unequal family sizes**, because a few sires are used heavily.
- **A trait recorded on one sex only.** `yield` is missing for males, as a milk trait would be.
- **Missing genotype calls**, at about one per cent, as any chip has.
- **Markers on ten chromosomes** with irregular spacing, so subsetting by genomic region is meaningful.

Because the data are simulated, the true breeding values are known and are kept in `colData`, which makes the object useful for checking that a prediction method recovers what it should.

Source

Simulated by `data-raw/make_demo.R` in the package sources.

See Also

`simulateBreeding()` to generate your own, `Hmatrix()` for the analysis this data set is shaped for.

Examples

```
data(demoBreeding)
demoBreeding

# 60 of the 180 individuals in the pedigree were never genotyped
nrow(pedigree(demoBreeding)) - ncol(demoBreeding)

# the single-step matrix covers all of them
H <- Hmatrix(demoBreeding)
dim(H)
```

Dmatrix *Dominance genomic relationship matrix*

Description

Builds the dominance relationship matrix from marker dosages using the genotypic parameterisation of Vitezica, Varona and Legarra (2013). Heterozygotes and the two homozygotes are coded so that the resulting matrix captures dominance deviations, orthogonal to the additive relationships returned by `Gmatrix()`.

Usage

```
Dmatrix(x, ...)

## S4 method for signature 'BreedingExperiment'
Dmatrix(x, min_maf = 0, impute = TRUE, ...)
```

Arguments

`x`	A BreedingExperiment object.
`...`	Unused.
`min_maf`	Markers with a minor allele frequency below this are dropped.
`impute`	Allow missing dosages, giving them a dominance contribution of zero. If `FALSE`, missing values raise an error.

Details

Fitting additive and dominance effects together lets a model separate the part of the genetic variance that is inherited predictably from the part that is not, which matters for crossbreeding and for predicting the performance of specific crosses.

Value

A symmetric numeric matrix over the genotyped individuals.

Dosages that are not whole numbers

Vitezica and others code the genotypes AA, Aa and aa as $-2q^2$, $2pq$ and $-2p^2$. Those three values lie on a single quadratic in the dosage g,

$$-g^2 + (2p + 1)g - 2p^2,$$

which is what this function evaluates. At dosages 2, 1 and 0 it returns the three published values exactly, and an expected dosage from imputation falls between them rather than being forced into one genotype class.

A missing call is given a dominance contribution of zero, which is the weight that mean imputation already gives it in `Gmatrix()`.

References

Vitezica, Z. G., Varona, L. & Legarra, A. (2013) "On the additive and dominant variance and covariance of individuals within the genomic selection scope." Genetics 195, 1223-1230. doi:10.1534/genetics.113.155176

See Also

`Gmatrix()`

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 20, n_marker = 200)
D <- Dmatrix(be)
round(D[1:4, 1:4], 3)
```

`filterMarkers` *Filter markers on quality*

Description

Removes markers that are uninformative or poorly genotyped: those below a minor allele frequency threshold, those called in too few individuals, and optionally those that are monomorphic.

Usage

```
filterMarkers(x, ...)

## S4 method for signature 'BreedingExperiment'
filterMarkers(
  x,
  min_maf = 0.01,
  min_call_rate = 0.9,
  drop_monomorphic = TRUE,
  ...
)
```

Arguments

x	A BreedingExperiment object.
...	Unused.
min_maf	Minimum minor allele frequency to keep.
min_call_rate	Minimum proportion of individuals with a call.
drop_monomorphic	Drop markers with no variation.

Value

A BreedingExperiment object with fewer rows.

Choosing the thresholds

`min_maf` and `min_call_rate` are judgement calls, not constants of nature. Common starting points are a minor allele frequency of 0.01 to 0.05 and a call rate of 0.90 to 0.95. Raise `min_maf` when the panel is large and the sample small, because a rare allele seen in one or two individuals carries almost no information about relationships but still adds noise; lower it when rare variants are the point of the study. Run this before `imputeMarkers()`, so that imputation is not asked to rescue markers that should simply be discarded.

See Also

`imputeMarkers()`, `alleleFrequency()`

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 30, n_marker = 100, missing = 0.05)
nrow(be)
nrow(filterMarkers(be, min_maf = 0.05, min_call_rate = 0.9))
```

---

genotyped *Which individuals are genotyped*

Description

Returns the identifiers that appear in the genotype matrix. In a single-step analysis the pedigree usually names more individuals than this, and the difference is what `Hmatrix()` reconciles.

Usage

```
genotyped(x, ...)

## S4 method for signature 'BreedingExperiment'
genotyped(x, ...)
```

Arguments

x	A BreedingExperiment object.
...	Unused.

Value

A character vector of individual identifiers.

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 8, n_marker = 20)
genotyped(be)
```

---

genotypes *Genotype dosages*

Description

Gets or sets the matrix of allele dosages, a shorthand for the genotype assay. Markers are rows and individuals are columns.

Usage

```
genotypes(x, ...)

genotypes(x) <- value

## S4 method for signature 'BreedingExperiment'
genotypes(x, ...)

## S4 replacement method for signature 'BreedingExperiment'
genotypes(x) <- value
```

Arguments

x	A BreedingExperiment object.
...	Unused.
value	A numeric matrix of dosages coded 0, 1 or 2.

Value

genotypes() returns a numeric matrix; the replacement form returns the updated object.

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 6, n_marker = 10)
genotypes(be)[1:5, 1:4]
```

`Gmatrix` *Genomic relationship matrix*

Description

Builds the additive genomic relationship matrix G from marker dosages using VanRaden's first method: with W the marker matrix centred by twice the allele frequency, $G = WW'/2\sum p(1-p)$.

Usage

```
Gmatrix(x, ...)

## S4 method for signature 'BreedingExperiment'
Gmatrix(x, min_maf = 0, impute = TRUE, ...)
```

Arguments

`x`	A BreedingExperiment object.
`...`	Unused.
`min_maf`	Markers with a minor allele frequency below this are dropped.
`impute`	Replace missing dosages by twice the allele frequency before computing, which gives a missing call no weight in the result. If `FALSE`, missing values raise an error.

Details

G plays the same role as the pedigree matrix A but is estimated from markers, so it captures Mendelian sampling that a pedigree cannot: full sibs have an expected pedigree relationship of exactly 0.5, whereas their genomic relationship varies around it.

Value

A symmetric numeric matrix over the genotyped individuals.

References

VanRaden, P. M. (2008) "Efficient methods to compute genomic predictions." Journal of Dairy Science 91, 4414-4423. doi:10.3168/jds.20070980

See Also

`Amatrix()`, `Dmatrix()`, `Hmatrix()`

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 20, n_marker = 200)
G <- Gmatrix(be)
round(G[1:4, 1:4], 3)
```

`Hmatrix` *Single-step relationship matrix*

Description

Builds the matrix H that combines pedigree and genomic information, so that genotyped and ungenotyped individuals can be analysed in a single model. This is the relationship structure behind single-step genomic prediction (ssGBLUP).

Usage

```
Hmatrix(x, ...)

## S4 method for signature 'BreedingExperiment'
Hmatrix(x, blend = 0.05, tune = TRUE, min_maf = 0, impute = TRUE, ...)
```

Arguments

`x`	A BreedingExperiment object with both a pedigree and genotypes.
`...`	Unused.
`blend`	Weight w given to A_{22} when blending, between 0 and 1. The default, 0.05, is a common choice.
`tune`	Rescale G to the scale of A_{22} before blending.
`min_maf`, `impute`	Passed to `Gmatrix()`.

Details

Real breeding programmes genotype only part of the population. Using only G throws away every ungenotyped relative; using only A throws away the markers. H keeps both: relationships among genotyped individuals come from the markers, relationships among ungenotyped individuals are those from the pedigree corrected by what the markers reveal about their genotyped relatives, and the two groups are connected through the pedigree.

Writing 1 for ungenotyped and 2 for genotyped individuals,

$$H_{22} = G^*$$

$$H_{12} = A_{12}A_{22}^{-1}G^*$$

$$H_{11} = A_{11} + A_{12}A_{22}^{-1}(G^* - A_{22})A_{22}^{-1}A_{21}$$

Because G and A_{22} are estimated on different scales, G is first made compatible with A_{22} in two standard steps. *Tuning* rescales G so that its mean diagonal and mean off-diagonal match those of A_{22}. *Blending* then mixes in a small share of A_{22}, $G^* = (1 - w)G + wA_{22}$, which also makes the result invertible when markers are fewer than individuals.

Value

A symmetric numeric matrix over every individual in the pedigree, ungenotyped individuals first, with an attribute `genotyped` giving the identifiers of the genotyped group. Genotyped individuals that are absent from the pedigree cannot be placed in H and are dropped with a warning.

Choosing blend and tune

Leave `tune = TRUE` unless you have already put G on the scale of A_{22} yourself. Untuned, the two matrices usually disagree about the average level of relationship, and that disagreement is passed straight into the ungenotyped block, where it shows up as breeding values that are biased between the genotyped and ungenotyped groups.

`blend` has two jobs. A small weight on A_{22} makes G^* invertible, which matters whenever there are fewer markers than genotyped individuals, and it pulls the markers back towards the pedigree where the marker estimate is noisy. Values between 0.01 and 0.10 are usual and 0.05 is a reasonable default; 0.05 to 0.10 suits a small marker panel, and 0.01 to 0.02 a dense one on many individuals. `blend = 0` keeps G as it is and will fail if G is singular. `blend = 1` discards the markers altogether and returns the pedigree matrix, which is useful only as the check shown in the examples.

References

Legarra, A., Aguilar, I. & Misztal, I. (2009) "A relationship matrix including full pedigree and genomic information." Journal of Dairy Science 92, 4656-4663. doi:10.3168/jds.20092061

Christensen, O. F. & Lund, M. S. (2010) "Genomic prediction when some animals are not genotyped." Genetics Selection Evolution 42, 2. doi:10.1186/12979686422

Aguilar, I., Misztal, I., Johnson, D. L., Legarra, A., Tsuruta, S. & Lawlor, T. J. (2010) "Hot topic: A unified approach to utilize phenotypic, full pedigree, and genomic information for genetic evaluation of Holstein final score." Journal of Dairy Science 93, 743-752. doi:10.3168/jds.20092730

See Also

`Amatrix()`, `Gmatrix()`

Examples

```
# only some of the individuals are genotyped
set.seed(1)
be <- simulateBreeding(n_ind = 30, n_marker = 300, genotyped = 20)
H <- Hmatrix(be)
dim(H)
attr(H, "genotyped")
```

`imputeMarkers` *Impute missing genotypes*

Description

Fills missing dosages with twice the allele frequency of the marker, the simple mean imputation that relationship matrices assume. It is adequate for the low rates of missingness left after quality control; for anything more, impute with dedicated software before building the object.

Usage

```
imputeMarkers(x, ...)

## S4 method for signature 'BreedingExperiment'
imputeMarkers(x, drop_uncalled = TRUE, ...)
```

Arguments

`x`	A BreedingExperiment object.
`...`	Unused.
`drop_uncalled`	Drop markers with no genotype calls at all, with a warning. If `FALSE`, their presence is an error.

Details

Mean imputation leaves an expected dosage rather than a whole number of copies, which the container accepts. Note what that costs: an imputed cell is no longer a genotype call, and its value alone no longer says so, since a marker whose mean dosage is exactly 1 imputes to exactly 1. The cells that were filled in are therefore recorded in a logical assay named `imputed`, reachable with `assay(x, "imputed")` and carried through subsetting with everything else. `alleleFrequency()` consults it, so observed heterozygosity is unchanged by imputing. `callRate`, on the other hand, counts dosages that are present and so becomes 1 for every marker; record call rates before imputing if you need them.

A marker with no calls at all has no frequency to impute from. Such markers are dropped with a warning rather than being filled in, because filling them would invent a genotype for every individual at that marker. `filterMarkers()` removes them too, and running it first is the better habit.

Value

A BreedingExperiment object with no missing genotypes and an `imputed` assay marking the cells that were filled in.

See Also

`filterMarkers()`, `alleleFrequency()`

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 20, n_marker = 50, missing = 0.1)
anyNA(genotypes(be))
anyNA(genotypes(imputeMarkers(be)))
```

`inbreeding` *Inbreeding coefficients*

Description

Returns the inbreeding coefficient of each individual, obtained from the diagonal of the pedigree relationship matrix as $F_i = A_{ii} - 1$.

Usage

```
inbreeding(x, ...)

## S4 method for signature 'BreedingExperiment'
inbreeding(x, ...)
```

Arguments

`x`	A BreedingExperiment object.
`...`	Unused.

Value

A named numeric vector of inbreeding coefficients.

See Also

`Amatrix()`

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 12, n_marker = 30)
round(inbreeding(be), 4)
```

pedigree *Pedigree of a breeding experiment*

Description

Gets or sets the pedigree, a table with columns `id`, `sire` and `dam`. The pedigree may include ancestors that were never genotyped; those individuals are what allow `Hmatrix()` to combine pedigree and genomic information.

Usage

```
pedigree(x, ...)

pedigree(x) <- value

## S4 method for signature 'BreedingExperiment'
pedigree(x, ...)

## S4 replacement method for signature 'BreedingExperiment'
pedigree(x) <- value
```

Arguments

`x`	A BreedingExperiment object.
`...`	Unused.
`value`	A data frame with columns `id`, `sire` and `dam`.

Value

`pedigree()` returns a S4Vectors::DataFrame; the replacement form returns the updated object.

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 8, n_marker = 20)
head(pedigree(be))
```

simulateBreeding *Simulate a small breeding population*

Description

Generates a pedigreed population with markers and a heritable trait, for examples, tests and teaching. Founders are unrelated; later generations are produced by mating sampled parents, and each offspring inherits one allele from each parent at every marker, so the simulated genotypes and the pedigree agree with each other.

Usage

```
simulateBreeding(
  n_ind = 50,
  n_marker = 200,
  n_founder = 10,
  genotyped = NULL,
  n_qtl = 20,
  h2 = 0.4,
  missing = 0
)
```

Arguments

n_ind	Number of individuals in total.
n_marker	Number of markers.
n_founder	Number of unrelated founders. Capped at n_ind; at least two are needed if there are to be any offspring.
genotyped	Number of individuals with genotypes, taken from the most recent ones. Defaults to all of them.
n_qtl	Number of markers with an effect on the trait.
h2	Narrow-sense heritability of the simulated trait, between 0 and 1.
missing	Proportion of genotype calls set to NA, between 0 and 1.

Details

Set genotyped below n_ind to leave some individuals ungenotyped, which is the situation Hmatrix() is designed for. The genotyped individuals are the most recent ones, so the ungenotyped are the earlier generations, as in a programme that began genotyping part way through.

Note that n_founder defaults to 10 and is capped at n_ind, so a call such as simulateBreeding(n_ind = 8) returns eight unrelated founders and no pedigree structure at all. Ask for more individuals than founders to get relatives.

The trait is scaled so that the genetic variance is h2 and the residual variance is 1 - h2, and trueBV in the column data is the genetic component on that scale.

The function draws random numbers but does not set the seed itself. Call set.seed() beforehand if you need the same population twice.

Value

A BreedingExperiment object with a phenotype column and the true breeding value in colData.

See Also

BreedingExperiment(), Hmatrix()

Examples

```
set.seed(1)
be <- simulateBreeding(n_ind = 25, n_marker = 100)
be
head(as.data.frame(colData(be)))

# a population where only the recent individuals were genotyped
part <- simulateBreeding(n_ind = 30, n_marker = 100, n_founder = 6,
                         genotyped = 20)
c(pedigree = nrow(pedigree(part)), genotyped = ncol(part))
```

`sortPedigree` *Order a pedigree so that parents precede their offspring*

Description

The tabular method for building a relationship matrix requires that every individual appears after both of its parents. This performs that topological sort and reports a cycle if the pedigree contains one.

Usage

```
sortPedigree(ped)
```

Arguments

`ped`	A data frame or S4Vectors::DataFrame with columns `id`, `sire` and `dam`.

Value

A data frame with the same columns, reordered.

Examples

```
ped <- data.frame(id = c("c", "a", "b"),
                  sire = c("a", NA, NA),
                  dam = c("b", NA, NA))
sortPedigree(ped)
```

Index

∗ **datasets**
demoBreeding, 8

alleleFrequency, 2
alleleFrequency(), *11*, *16*
alleleFrequency,BreedingExperiment-method *(*alleleFrequency*)*, 2
Amatrix, 4
Amatrix(), *6, 7*, *13*, *15*, *17*
Amatrix,BreedingExperiment-method *(*Amatrix*)*, 4

BreedingExperiment, *3, 4*, 5, *5, 6*, *8, 9*, *11–14*, *16–19*
BreedingExperiment(), *7*, *19*
BreedingExperiment-class, 6

demoBreeding, 8
Dmatrix, 9
Dmatrix(), *13*
Dmatrix,BreedingExperiment-method *(*Dmatrix*)*, 9

filterMarkers, 10
filterMarkers(), *3*, *16*
filterMarkers,BreedingExperiment-method *(*filterMarkers*)*, 10

GenomicRanges::GRanges, *5*, *7*
genotyped, 11
genotyped,BreedingExperiment-method *(*genotyped*)*, 11
genotypes, 12
genotypes,BreedingExperiment-method *(*genotypes*)*, 12
genotypes<- *(*genotypes*)*, 12
genotypes<-,BreedingExperiment-method *(*genotypes*)*, 12
Gmatrix, 13
Gmatrix(), *4*, *6, 7*, *9, 10*, *14, 15*
Gmatrix,BreedingExperiment-method *(*Gmatrix*)*, 13

Hmatrix, 14
Hmatrix(), *4–8*, *11*, *13*, *18, 19*
Hmatrix,BreedingExperiment-method *(*Hmatrix*)*, 14

imputeMarkers, 16
imputeMarkers(), *3*, *11*
imputeMarkers,BreedingExperiment-method *(*imputeMarkers*)*, 16
inbreeding, 17
inbreeding(), *4*
inbreeding,BreedingExperiment-method *(*inbreeding*)*, 17

pedigree, 18
pedigree,BreedingExperiment-method *(*pedigree*)*, 18
pedigree<- *(*pedigree*)*, 18
pedigree<-,BreedingExperiment-method *(*pedigree*)*, 18

S4Vectors::DataFrame, *3*, *5*, *7*, *18*, *20*
set.seed(), *19*
show,BreedingExperiment-method *(*BreedingExperiment-class*)*, 6
simulateBreeding, 18
simulateBreeding(), *6*, *8*
sortPedigree, 20
SummarizedExperiment::RangedSummarizedExperiment, *7*
SummarizedExperiment::SummarizedExperiment(), *5*