Package ‘visPedigree’

July 15, 2026

Type Package

Title Tidying, Analysis, and Fast Visualization of Animal and Plant Pedigrees

Version 1.9.0

Description Provides tools for the analysis and visualization of animal and plant pedigrees. Analytical methods include equivalent complete generations, generation intervals, effective population size (via inbreeding, coancestry, and demographic approaches), founder and ancestor contributions, partial inbreeding, genetic diversity indices, and additive (A), dominance (D), and epistatic (AA) relationship matrices. Core algorithms — ancestry tracing, topological sorting, inbreeding coefficients, and matrix construction — are implemented in C++ ('Rcpp', 'RcppArmadillo') and 'data.table', scaling to pedigrees with over one million individuals. Matrix-free pedigree traversals apply additive relationship matrices or their inverses to vectors and matrices without materializing dense square matrices. Pedigree graphs are rendered via 'igraph' with support for compact full-sib family display; relationship matrices can be visualized as heatmaps. Supports complex mating systems, including selfing and pedigrees in which the same individual can appear as both sire and dam.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports data.table (>= 1.14.0), igraph (>= 1.3.0), Matrix, methods, Rcpp, lattice

LinkingTo Rcpp, RcppArmadillo

Suggests nadiv (>= 2.18.0), testthat (>= 3.0.0), knitr, rmarkdown, devtools, roxygen2, pkgdown

URL https://github.com/luansheng/visPedigree, https://luansheng.github.io/visPedigree/

BugReports https://github.com/luansheng/visPedigree/issues

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.3

NeedsCompilation yes

Author Sheng Luan [aut, cre]

Maintainer Sheng Luan <luansheng@gmail.com>

Repository CRAN

Date/Publication 2026-07-15 05:30:02 UTC

Contents

as_tidyped . . . 3
big_family_size_ped . . . 4
complex_ped . . . 5
deep_ped . . . 5
expand_pedmat . . . 6
half_founder_ped . . . 7
has_candidates . . . 7
has_inbreeding . . . 8
inbred_ped . . . 8
inbreed . . . 9
is_tidyped . . . 10
loop_ped . . . 10
pedancestry . . . 11
pedcontrib . . . 12
pedecg . . . 14
pedfclass . . . 15
pedgenint . . . 16
pedhalflife . . . 18
pediv . . . 20
pedmat . . . 22
pedmeta . . . 25
pedne . . . 26
pedpartial . . . 28
pedprod . . . 29
pedrel . . . 31
pedstats . . . 33
pedsubpop . . . 35
plot.tidyped . . . 36
print.pedcontrib . . . 36
print.pediv . . . 37
print.pedstats . . . 37
print.summary.tidyped . . . 38
print.tidyped . . . 38
query_relationship . . . 39

simple_ped 40
small_ped 40
splitped 41
summary.tidyped 42
summary_pedmat 43
tidyped 44
vismat 46
visped 50
[.tidyped 55

Index **57**

`as_tidyped` *Restore the tidyped class to a manipulated pedigree*

Description

Rapidly restores the `tidyped` class to a `data.table` or `data.frame` that was previously processed by `tidyped()` but lost its class attributes due to data manipulation.

Usage

```
as_tidyped(x)
```

Arguments

`x`	A `data.table` or `data.frame` that was previously a tidyped object. It must still contain the core columns: `Ind`, `Sire`, `Dam`, `Sex`, `Gen`, `IndNum`, `SireNum`, `DamNum`.

Details

This is a lightweight operation that only checks for the required columns and re-attaches the class—it does **not** re-run the full pedigree sorting, generation inference, or loop detection.

This helper is intended for objects that still contain the core pedigree columns and numeric indices, but no longer inherit from `tidyped`. A common reproducible case is `rbind()` on two `tidyped` fragments, which typically returns a plain `data.table`. Converting a `tidyped` object to a plain `data.frame` and then subsetting it also drops the class.

Some operations, such as `merge()` or certain dplyr workflows, may or may not preserve the `tidyped` class depending on the versions of **data.table**, **dplyr**, and the exact method dispatch path used in the current R session. Therefore, `as_tidyped()` should be viewed as a safe recovery helper rather than something only needed after one specific verb.

Typical class-loss scenarios include:

- `rbind(tped1, tped2)` — often returns plain `data.table`
- `as.data.frame(tped)[rows, ]` — returns plain `data.frame`
- manual class removal or serialization / import workflows

After such operations, downstream analysis functions (e.g., pedstats, pedne) will either error or automatically restore the class. You can also call as_tidyped() explicitly to restore the class yourself.

Value

A tidyped object.

See Also

tidyped, new_tidyped

Examples

```
library(visPedigree)
tp <- tidyped(simple_ped)
class(tp)
# [1] "tidyped"    "data.table" "data.frame"

# Simulate class loss via rbind()
tp2 <- rbind(tp[1:5], tp[6:10])
class(tp2)
# [1] "data.table" "data.frame"

# Restore the class
tp3 <- as_tidyped(tp2)
class(tp3)
# [1] "tidyped"    "data.table" "data.frame"

# It can also restore from a plain data.frame if core columns are intact
tp_df <- as.data.frame(tp)
tp4 <- tp_df[tp_df$Gen > 1, ]
class(tp4)
# [1] "data.frame"

tp5 <- as_tidyped(tp4)
class(tp5)
# [1] "tidyped"    "data.table" "data.frame"
```

---

big_family_size_ped *A large pedigree with big family sizes*

Description

A dataset containing a pedigree with many full-sib individuals per family.

Usage

```
big_family_size_ped
```

Format

A data.table with 8 columns:

Ind Individual ID
Sire Sire ID
Dam Dam ID
Sex Sex of the individual
Year Year of birth
IndNum Numeric ID for individual
SireNum Numeric ID for sire
DamNum Numeric ID for dam

---

complex_ped *A complex pedigree*

Description

A dataset containing a large, complex pedigree covering about 100 generations, useful for testing the performance and accuracy of partial inbreeding and similar calculations.

Usage

```
complex_ped
```

Format

A data.table with a standard pedigree structure.

---

deep_ped *A deep pedigree*

Description

A dataset containing a pedigree with many generations.

Usage

```
deep_ped
```

Format

A data.table with 4 columns:

Ind Individual ID
Sire Sire ID
Dam Dam ID
Sex Sex of the individual

expand_pedmat — *Expand a Compact Pedigree Matrix to Full Dimensions*

Description

Restores a compact pedmat to its original dimensions by mapping each individual to their family representative's values. For non-compact matrices, returns the matrix unchanged.

Usage

```
expand_pedmat(x)
```

Arguments

x	A pedmat object from pedmat.

Details

For compact matrices, full-siblings within the same family will have identical relationship values in the expanded matrix because they shared the same representative during calculation.

Value

Matrix or vector with original pedigree dimensions:

- Matrices: Row and column names set to all individual IDs
- Vectors (e.g., method="f"): Names set to all individual IDs

The result is **not** a pedmat object (S3 class stripped).

See Also

pedmat, query_relationship

Examples

```
tped <- tidyped(small_ped)

# Compact matrix
A_compact <- pedmat(tped, method = "A", compact = TRUE)
dim(A_compact)  # Reduced dimensions

# Expand to full size
A_full <- expand_pedmat(A_compact)
dim(A_full)  # Original dimensions restored

# Non-compact matrices are returned unchanged
A <- pedmat(tped, method = "A", compact = FALSE)
A2 <- expand_pedmat(A)
```

```
identical(dim(A), dim(A2))  # TRUE
```

`half_founder_ped` *A pedigree with half founders*

Description

A dataset from ENDOG containing individuals with a single missing parent (half founders). Useful for testing genetic algorithms correctly conserving probability mass for missing lineages.

Usage

```
half_founder_ped
```

Format

A data.frame with 4 columns:

Ind Individual ID

Sire Sire ID

Dam Dam ID

Sex Sex of the individual

`has_candidates` *Check whether a tidyped object contains candidate flags*

Description

Check whether a tidyped object contains candidate flags

Usage

```
has_candidates(x)
```

Arguments

x	A tidyped object.

Value

Logical scalar.

has_inbreeding — *Check whether a tidyped object contains inbreeding coefficients*

Description

Check whether a tidyped object contains inbreeding coefficients

Usage

```
has_inbreeding(x)
```

Arguments

x	A tidyped object.

Value

Logical scalar.

inbred_ped — *A highly inbred pedigree*

Description

A simulated pedigree designed to demonstrate high levels of inbreeding and partial inbreeding decomposition. Contains full-sib mating and backcrossing.

Usage

```
inbred_ped
```

Format

A data.table with 5 columns:

Ind Individual ID

Sire Sire ID

Dam Dam ID

Sex Sex of the individual

Gen Generation number

`inbreed` *Calculate inbreeding coefficients*

Description

`inbreed` function calculates the inbreeding coefficients for all individuals in a tidied pedigree.

Usage

```
inbreed(ped, ...)
```

Arguments

`ped`	A `tidyped` object.
`...`	Additional arguments (currently ignored).

Details

This function takes a pedigree tidied by the `tidyped` function and calculates the inbreeding coefficients using an optimized C++ implementation of the Sargolzaei & Iwaisaki (2005) LAP (Longest Ancestral Path) bucket algorithm. This method is the fastest known direct algorithm for computing all inbreeding coefficients: it replaces the O(N^2) linear scan of Meuwissen & Luo (1992) with O(1) bucket pops and selective ancestor clearing, giving $O(\sum m_i)$ total work where m_i is the number of distinct ancestors of individual i. At $N = 1{,}000{,}000$, the kernel completes in approximately 0.12 s — over 10× faster than the previous Meuwissen & Luo (1992) implementation and on par with the **pedigreemm** reference C implementation of the same algorithm. It is the core engine used by both `tidyped(..., inbreed = TRUE)` and `pedmat(..., method = "f")`, ensuring consistent results across the package.

Value

A `tidyped` object with an additional column **f**.

Examples

```
library(visPedigree)
data(simple_ped)
ped <- tidyped(simple_ped)
ped_f <- inbreed(ped)
ped_f[f > 0, .(Ind, Sire, Dam, f)]
```

is_tidyped — *Test if an object is a tidyped*

Description

Test if an object is a tidyped

Usage

```
is_tidyped(x)
```

Arguments

x	An object to test.

Value

Logical scalar.

loop_ped — *A pedigree with loops*

Description

A dataset containing a pedigree with circular mating loops.

Usage

```
loop_ped
```

Format

A data.table with 3 columns:

Ind Individual ID

Sire Sire ID

Dam Dam ID

pedancestry *Calculate Ancestry Proportions*

Description

Estimates the proportion of genes for each individual that originates from specific founder groups (e.g., breeds, source populations).

Usage

```
pedancestry(ped, foundervar, target_labels = NULL)
```

Arguments

ped	A tidyped object.
foundervar	Character. The name of the column containing founder-group labels (e.g., "Breed", "Origin").
target_labels	Character vector. Specific founder-group labels to track. If NULL, all unique labels in foundervar among founders are used.

Value

A data.table with columns:

- Ind: Individual ID.
- One column per tracked label (named after each unique value in foundervar among founders, or as specified by target_labels). Each value gives the proportion of genes (0–1) originating from that founder group. Row sums across all label columns equal 1.

Examples

```
library(data.table)
# Create dummy labels for founders
tp <- tidyped(small_ped)
tp_dated <- copy(tp)
founders <- tp_dated[is.na(Sire) & is.na(Dam), Ind]
# Assign 'LineA' and 'LineB'
tp_dated[Ind %in% founders[1:(length(founders)/2)], Origin := "LineA"]
tp_dated[is.na(Origin), Origin := "LineB"]

# Calculate ancestry proportions for all individuals
anc <- pedancestry(tp_dated, foundervar = "Origin")
print(tail(anc))
```

pedcontrib *Calculate Founder and Ancestor Contributions*

Description

Calculates genetic contributions from founders and influential ancestors. Implements the gene dropping algorithm for founder contributions and Boichard's algorithm for ancestor contributions to estimate the effective number of founders (f_e) and ancestors (f_a).

Usage

```
pedcontrib(
  ped,
  reference = NULL,
  mode = c("both", "founder", "ancestor"),
  top = 20
)
```

Arguments

`ped`	A `tidyped` object.
`reference`	Character vector. Optional subset of individual IDs defining the reference population. If NULL, uses all individuals in the most recent generation.
`mode`	Character. Type of contribution to calculate: • `"founder"`: Founder contributions (f_e). • `"ancestor"`: Ancestor contributions (f_a). • `"both"`: Both founder and ancestor contributions.
`top`	Integer. Number of top contributors to return. Default is 20.

Details

Founder Contributions (f_e): Calculated by probabilistic gene flow from founders to the reference cohort. When individual ancestors with one unknown parent exist, "phantom" parents are temporarily injected correctly conserving the probability mass.

Ancestor Contributions (f_a): Calculated using Boichard's iterative algorithm (1997), accounting for:

- Marginal genetic contribution of each ancestor
- Long-term contributions through multiple pathways

The parameter f_a acts as a stringent metric since it identifies the bottlenecks of genetic variation in pedigrees.

Value

A list with class `pedcontrib` containing:

- `founders`: A `data.table` of founder contributions (if mode includes "founder", or "both").
- `ancestors`: A `data.table` of ancestor contributions (if mode includes "ancestor", or "both").
- `summary`: A `list` of summary statistics including:
  - `f_e`: Classical effective number of founders ($q = 2$, Lacy 1989).
  - `f_e_H`: Information-theoretic effective number of founders ($q = 1$, Shannon entropy): $f_e^{(H)} = \exp(-\sum p_i \ln p_i)$.
  - `f_a`: Classical effective number of ancestors ($q = 2$, Boichard 1997).
  - `f_a_H`: Information-theoretic effective number of ancestors ($q = 1$): $f_a^{(H)} = \exp(-\sum q_k \ln q_k)$.

Each contribution table contains:

- `Ind`: Individual ID.
- `Contrib`: Contribution to the reference population (0-1).
- `CumContrib`: Cumulative contribution.
- `Rank`: Rank by contribution.

References

Boichard, D., Maignel, L., & Verrier, É. (1997). The value of using probabilities of gene origin to measure genetic variability in a population. Genetics Selection Evolution, 29(1), 5-23.

Examples

```
library(data.table)
# Load a sample pedigree
tp <- tidyped(small_ped)

# Calculate both founder and ancestor contributions for reference population
ref_ids <- c("Z1", "Z2", "X", "Y")
contrib <- pedcontrib(tp, reference = ref_ids, mode = "both")

# Print results including f_e, f_e(H), f_a, and f_a(H)
print(contrib)

# Access Shannon-entropy effective numbers directly
contrib$summary$f_e_H   # Information-theoretic effective founders (q=1)
contrib$summary$f_e     # Classical effective founders (q=2)
contrib$summary$f_a_H   # Information-theoretic effective ancestors (q=1)
contrib$summary$f_a     # Classical effective ancestors (q=2)

# Diversity ratio rho > 1 indicates long-tail founder value
contrib$summary$f_e_H / contrib$summary$f_e
```

pedecg *Calculate Equi-Generate Coefficient*

Description

Estimates the number of distinct ancestral generations using the Equi-Generate Coefficient (ECG). The ECG is calculated as 1/2 of the sum of the parents' ECG values plus 1.

Usage

```
pedecg(ped)
```

Arguments

ped	A `tidyped` object.

Value

A `data.table` with columns:

- `Ind`: Individual ID.
- `ECG`: Equi-Generate Coefficient.
- `FullGen`: Fully traced generations (depth of complete ancestry).
- `MaxGen`: Maximum ancestral generations (depth of deepest ancestor).

References

Boichard, D., Maignel, L., & Verrier, E. (1997). The value of using probabilities of gene origin to measure genetic variability in a population. Genetics Selection Evolution, 29(1), 5.

Examples

```
tp <- tidyped(simple_ped)
ecg <- pedecg(tp)

# ECG combines pedigree depth and completeness
head(ecg)

# Individuals with deeper and more complete ancestry have larger ECG values
ecg[order(-ECG)][1:5]
```

pedfclass *Summarize Inbreeding Levels*

Description

Classifies individuals into inbreeding levels based on their inbreeding coefficients (F) according to standard or user-defined thresholds.

Usage

```
pedfclass(ped, breaks = c(0.0625, 0.125, 0.25), labels = NULL)
```

Arguments

`ped`	A `tidyped` object.
`breaks`	Numeric vector of strictly increasing positive upper bounds for inbreeding classes. Default is `c(0.0625, 0.125, 0.25)`, corresponding approximately to half-sib, avuncular/grandparent, and full-sib/parent-offspring mating thresholds. The class `"F = 0"` is always kept as a fixed first level. A final open-ended class `"F > max(breaks)"` is always appended automatically.
`labels`	Optional character vector of interval labels. If `NULL`, labels are generated automatically from `breaks`. When supplied, its length must equal `length(breaks)`, with each element naming the bounded interval `(breaks[i-1], breaks[i]]`. The open-ended tail class is always auto-generated and cannot be overridden.

Details

The default thresholds follow common pedigree interpretation rules:

- `F = 0.0625`: approximately the offspring of half-sib mating.
- `F = 0.125`: approximately the offspring of avuncular or grandparent-grandchild mating.
- `F = 0.25`: approximately the offspring of full-sib or parent-offspring mating.

Therefore, assigning `F = 0.25` to the class `"0.125 < F <= 0.25"` is appropriate. If finer reporting is needed, supply custom `breaks`, for example to separate `0.25`, `0.375`, or `0.5`.

Value

A `data.table` with 3 columns:

`FClass` An ordered factor. By default it contains 5 levels: `"F = 0"`, `"0 < F <= 0.0625"`, `"0.0625 < F <= 0.125"`, `"0.125 < F <= 0.25"`, and `"F > 0.25"`. The number of levels equals `length(breaks) + 2` (the fixed zero class plus one class per bounded interval plus the open-ended tail).

`Count` Integer. Number of individuals in each class.

`Percentage` Numeric. Percentage of individuals in each class.

Examples

```
tp <- tidyped(simple_ped, addnum = TRUE)
pedfclass(tp)

# Finer custom classes (4 breaks, labels auto-generated)
pedfclass(tp, breaks = c(0.03125, 0.0625, 0.125, 0.25))

# Custom labels aligned to breaks (3 labels for 3 breaks; tail is auto)
pedfclass(tp, labels = c("Low", "Moderate", "High"))


tp_inbred <- tidyped(inbred_ped, addnum = TRUE)
pedfclass(tp_inbred)
```

---

pedgenint *Calculate Generation Intervals*

Description

Computes the generation intervals for the four gametic pathways: Sire to Son (SS), Sire to Daughter (SD), Dam to Son (DS), and Dam to Daughter (DD). The generation interval is defined as the age of the parents at the birth of their offspring.

Usage

```
pedgenint(
  ped,
  timevar = NULL,
  unit = c("year", "month", "day", "hour"),
  format = NULL,
  cycle = NULL,
  by = NULL
)
```

Arguments

`ped` A `tidyped` object.

`timevar` Character. The name of the column containing the **birth date** (or hatch date) of each individual. The column must be one of:

- `Date` or `POSIXct` (recommended).
- A date string parseable by `as.POSIXct` (e.g., `"2020-03-15"`). Use `format` for non-ISO strings.
- A numeric year (e.g., `2020`). Automatically converted to `Date` (`"YYYY-07-01"`) with a message. For finer precision, convert to `Date` beforehand.

If NULL, auto-detects columns named "BirthYear", "Year", "BirthDate", or "Date".

unit — Character. Output time unit for the interval: "year" (default), "month", "day", or "hour".

format — Character. Optional format string for parsing timevar when it contains non-standard date strings (e.g., "%d/%m/%Y" for "15/03/2020").

cycle — Numeric. Optional target (designed) length of one generation cycle expressed in units. When provided, an additional column GenEquiv is appended to the result, defined as:

$$GenEquiv_i = \frac{\bar{L}_i}{L_{cycle}}$$

where $\bar{L}_i$ is the observed mean interval for pathway i and L_{cycle} is cycle. A value > 1 means the observed interval exceeds the target cycle (lower breeding efficiency). Example: for Pacific white shrimp with a 180-day target cycle, set unit = "day", cycle = 180.

by — Character. Optional grouping column (e.g., "Breed", "Farm"). If provided, intervals are calculated within each group.

Details

Parent-offspring pairs with zero or negative intervals are excluded from the calculation because they typically indicate data entry errors or insufficient time resolution. If many zero intervals are expected (e.g., when using unit = "year" with annual spawners), consider using a finer time unit such as "month" or "day".

Numeric year columns (e.g., 2020) are automatically converted to Date by appending "-07-01" (mid-year) as a reasonable default. For more precise results, convert to Date before calling this function.

Value

A data.table with columns:

- Group: Grouping level (if by is used).
- Pathway: One of "SS", "SD", "DS", "DD", "SO", "DO", or "Average". SS/SD/DS/DD require offspring sex; SO (Sire-Offspring) and DO (Dam-Offspring) are computed from all parent-offspring pairs regardless of offspring sex.
- N: Number of parent-offspring pairs used.
- Mean: Average generation interval in unit.
- SD: Standard deviation of the interval.
- GenEquiv: (Optional) Generation equivalents based on cycle.

Examples

```
# ---- Basic usage with package dataset (numeric Year auto-converted) ----
tped <- tidyped(big_family_size_ped)
gi <- pedgenint(tped, timevar = "Year")
```

```
gi

# ---- Generation equivalents with cycle ----
gi2 <- pedgenint(tped, timevar = "Year", cycle = 2)
gi2
```

---

pedhalflife *Calculate Information-Theoretic Diversity Half-Life*

Description

Calculates the diversity half-life ($T_{1/2}$) of a pedigree across time points using a Renyi-2 entropy cascade framework. The total loss rate of genetic diversity is partitioned into three additive components:

- λ_e: foundation bottleneck (unequal founder contributions).
- λ_b: breeding bottleneck (overuse of key ancestors).
- λ_d: genetic drift (random sampling loss).

The function rolls over time points defined by `timevar`, computing f_e and f_a (via `pedcontrib`) and f_g (via the internal matrix-free coancestry engine) for each time point. Only the f_g statistic is requested; no redundant Ne calculations are performed.

Usage

```
pedhalflife(
  ped,
  timevar = "Gen",
  at = NULL,
  nsamples = 1000,
  ncores = 1,
  seed = NULL
)

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

## S3 method for class 'pedhalflife'
plot(x, type = c("log", "raw"), ...)
```

Arguments

`ped`	A `tidyped` object.
`timevar`	Character. Column name in `ped` that defines time points (e.g. `"Gen"`, `"Year"`). Default: `"Gen"`.

at	Optional vector of values selecting which time points to include (e.g., 2:4, 2010:2020, or non-consecutive c(2015, 2018, 2022)). Values must match entries in the timevar column. Non-numeric values are accepted but the OLS time axis will fall back to sequential indices. If NULL (default), all non-NA unique values in timevar are used.
nsamples	Integer. Sample size per time point for coancestry estimation (passed to the internal coancestry engine). Default: 1000.
ncores	Integer. Number of OpenMP threads for C++ backends. Default: 1.
seed	Integer or NULL. Random seed for reproducible coancestry sampling. Default: NULL.
x	A pedhalflife object.
...	Additional arguments (ignored).
type	Character. "log" for log-transformed values; "raw" for f_e, f_a, f_g.

Details

The mathematical identity underlying the cascade is:

$$\ln f_g = \ln f_e + \ln(f_a/f_e) + \ln(f_g/f_a)$$

Taking the negative time-slope of each term gives the λ components, which sum exactly by linearity of OLS:

$$\lambda_{total} = \lambda_e + \lambda_b + \lambda_d$$

$T_{1/2} = \ln 2/\lambda_{total}$ is the number of time-units (time points, years, generations) for diversity to halve.

Value

A list of class pedhalflife with two data.table components:

timeseries Per-time-point tracking with columns Time (time-point label from timevar), NRef, fe, fa, fg and their log transformations (lnfe, lnfa, lnfg, lnfafe, lnfgfa), plus TimeStep (numeric OLS time axis).

decay Single-row table with lambda_e, lambda_b, lambda_d, lambda_total, and thalf.

See Also

pediv, pedcontrib, tidyped

Examples

```
library(visPedigree)
data(simple_ped)
tp <- tidyped(simple_ped)

# 1. Calculate half-life using all available generations
hl <- pedhalflife(tp, timevar = "Gen")
print(hl)
```

```
# 2. View the underlying log-linear decay plot
plot(hl, type = "log")

# 3. Calculate half-life for a specific time window (e.g., Generations 2 to 4)
hl_subset <- pedhalflife(tp, timevar = "Gen", at = c(2, 3, 4))
print(hl_subset)
```

`pediv` *Calculate Genetic Diversity Indicators*

Description

Combines founder/ancestor contributions (f_e, f_a) and effective population size estimates (Ne) from three methods into a single summary object.

Usage

```
pediv(
  ped,
  reference = NULL,
  top = 20,
  nsamples = 1000,
  ncores = 1,
  seed = NULL
)
```

Arguments

`ped`	A `tidyped` object.
`reference`	Character vector. Optional subset of individual IDs defining the reference population. If NULL, uses all individuals in the most recent generation.
`top`	Integer. Number of top contributors to return in founder/ancestor tables. Default is 20.
`nsamples`	Integer. Number of individuals sampled per cohort for the coancestry Ne method and for f_g estimation. Very large cohorts are sampled down to this size to control memory usage (default: 1000).
`ncores`	Integer. Number of cores for parallel processing in the coancestry method. Default is 1.
`seed`	Integer or NULL. Random seed passed to `set.seed()` before sampling in the coancestry method, ensuring reproducible f_g and N_e estimates. Default is `NULL` (no fixed seed).

Details

Internally calls pedcontrib for f_e and f_a. The coancestry method is called via the internal calc_ne_coancestry() function directly so that f_g and the Ne estimate can be obtained from the same traced pedigree and the same batched matrix-free relationship products without duplication. The inbreeding and demographic Ne methods are obtained via pedne. All calculations use the same reference population. If any method fails (e.g., insufficient pedigree depth), its value is NA rather than stopping execution.

f_g (founder genome equivalents, Caballero & Toro 2000) is estimated from the diagonal-corrected mean coancestry of the reference population:

$$\hat{\bar{C}} = \frac{N-1}{N} \cdot \frac{\bar{a}_{off}}{2} + \frac{1+\bar{F}_s}{2N}$$

$$f_g = \frac{1}{2\hat{\bar{C}}}$$

where N is the full reference cohort size, $\bar{a}_{off}$ is the off-diagonal mean relationship among sampled individuals, and $\bar{F}_s$ is their mean inbreeding coefficient.

Value

A list with class pediv containing:

- summary: A single-row data.table with columns NRef, NFounder, feH, fe, NAncestor, faH, fa, fafe, fg, MeanCoan, GeneDiv, NSampledCoan, NeCoancestry, NeInbreeding, NeDemographic. Here feH and faH are the Shannon-entropy ($q = 1$) effective numbers of founders and ancestors, respectively. GeneDiv is the pedigree-based retained genetic diversity, computed as $1-\bar{C}$, where $\bar{C}$ is the diagonal-corrected population mean coancestry (MeanCoan).
- founders: A data.table of top founder contributions.
- ancestors: A data.table of top ancestor contributions.

See Also

pedcontrib, pedne, pedstats

Examples

```
tp <- tidyped(small_ped)
div <- pediv(tp, reference = c("Z1", "Z2", "X", "Y"), seed = 42L)
print(div)

# Access Shannon effective numbers from summary
div$summary$feH   # Shannon effective founders (q=1)
div$summary$faH   # Shannon effective ancestors (q=1)

# Founder diversity profile: NFounder >= feH >= fe
with(div$summary, c(NFounder = NFounder, feH = feH, fe = fe))
```

pedmat *Genetic Relationship Matrices and Inbreeding Coefficients*

Description

Optimized calculation of additive (A), dominance (D), epistatic (AA) relationship matrices, their inverses, and inbreeding coefficients (f). Uses Rcpp with Meuwissen & Luo (1992) algorithm for efficient computation.

Usage

```
pedmat(
  ped,
  method = "A",
  sparse = TRUE,
  invert_method = "auto",
  threads = 0,
  compact = FALSE
)
```

Arguments

`ped` A tidied pedigree from `tidyped`. Must be a single pedigree, not a splitped object. For splitped results, use `pedmat(ped_split$GP1, ...)` to process individual groups.

`method` Character, one of:

- `"A"`: Additive (numerator) relationship matrix (default)
- `"f"`: Inbreeding coefficients (returns named vector)
- `"Ainv"`: Inverse of A using Henderson's rules (O(n) complexity)
- `"D"`: Dominance relationship matrix
- `"Dinv"`: Inverse of D (requires matrix inversion)
- `"AA"`: Additive-by-additive epistatic matrix (A # A)
- `"AAinv"`: Inverse of AA

`sparse` Logical, if `TRUE` returns sparse Matrix (recommended for large pedigrees). Default is `TRUE`.

`invert_method` Character, method for matrix inversion (Dinv/AAinv only):

- `"auto"`: Auto-detect and use optimal method (default)
- `"sympd"`: Force Cholesky decomposition (faster for SPD matrices)
- `"general"`: Force general LU decomposition

`threads` Integer. Number of OpenMP threads to use. Use 0 to keep the system/default setting. Currently, multi-threading is explicitly implemented for:

- `"D"`: Dominance relationship matrix (significant speedup).
- `"Ainv"`: Inverse of A (only for large pedigrees, n >= 5000).

For "Dinv", "AA", and "AAinv", parallelism depends on the linked BLAS/LAPACK library (e.g., OpenBLAS, MKL, Accelerate) and is not controlled by this parameter. Methods "A" and "f" are single-threaded.

`compact` Logical, if TRUE compacts full-sibling families by selecting one representative per family. This dramatically reduces matrix dimensions for pedigrees with large full-sib groups. See Details.

Details

API Design:

Only a single method may be requested per call. This design prevents accidental heavy computations. If multiple matrices are needed, call `pedmat()` separately for each method.

Compact Mode (`compact = TRUE`):

Full-siblings share identical relationships with all other individuals. Compact mode exploits this by selecting one representative per full-sib family, dramatically reducing matrix size. For example, a pedigree with 170,000 individuals might compact to 1,800 unique relationship patterns.

Key features:

- `query_relationship(x, id1, id2)`: Query any individual pair, including merged siblings (automatic lookup)
- `expand_pedmat(x)`: Restore full matrix dimensions
- `vismat(x)`: Visualize directly (auto-expands compact)

Performance Notes:

- **Ainv**: O(n) complexity using Henderson's rules. Fast for any size.
- **Dinv/AAinv**: O(n^3) matrix inversion. Practical limits:
  - n < 500: ~10-20 ms
  - n = 1,000: ~40-60 ms
  - n = 2,000: ~130-150 ms
  - n > 2,000: Consider using `compact = TRUE`
- **Memory**: Sparse matrices use ~O(nnz) memory; dense use O(n^2)

Value

Returns a matrix or vector with S3 class "pedmat".

Object type by method:

- `method="f"`: Named numeric vector of inbreeding coefficients
- All other methods: Sparse or dense matrix (depending on `sparse`)

S3 Methods:

- `print(x)`: Display matrix with metadata header
- `summary_pedmat(x)`: Detailed statistics (size, compression, mean, density)
- `dim(x)`, `length(x)`, `Matrix::diag(x)`, `t(x)`: Standard operations

- `x[i, j]`: Subsetting (behaves like underlying matrix)
- `as.matrix(x)`: Convert to base matrix

Accessing Metadata (use `attr()`, not `$`):

- `attr(x, "ped")`: The pedigree used (or compact pedigree if `compact=TRUE`)
- `attr(x, "ped_compact")`: Compact pedigree (when `compact=TRUE`)
- `attr(x, "method")`: Calculation method used
- `attr(x, "call_info")`: Full calculation metadata (timing, sizes, etc.)
- `names(attributes(x))`: List all available attributes

Additional attributes when `compact = TRUE`:

- `attr(x, "compact_map")`: data.table mapping individuals to representatives
- `attr(x, "family_summary")`: data.table summarizing merged families
- `attr(x, "compact_stats")`: Compression statistics (ratio, n_families, etc.)

References

Meuwissen, T. H. E., & Luo, Z. (1992). Computing inbreeding coefficients in large populations. Genetics Selection Evolution, 24(4), 305-313.

Henderson, C. R. (1976). A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. Biometrics, 32(1), 69-83.

See Also

`tidyped` for preparing pedigree data, `pedprod` for matrix-free products with A or Ainv, `query_relationship` for querying individual pairs, `expand_pedmat` for restoring full dimensions, `vismat` for visualization, `inbreed` for simple inbreeding calculation

Examples

```
# Basic usage with small pedigree
library(visPedigree)
tped <- tidyped(small_ped)

# --- Additive Relationship Matrix (default) ---
A <- pedmat(tped)
A["A", "B"]      # Relationship between A and B
Matrix::diag(A)  # Diagonal = 1 + F (inbreeding)

# --- Inbreeding Coefficients ---
f <- pedmat(tped, method = "f")
f["Z1"]  # Inbreeding of individual Z1

# --- Using summary_pedmat() ---
summary_pedmat(A)   # Detailed matrix statistics

# --- Accessing Metadata ---
attr(A, "ped")              # Original pedigree
```

```
attr(A, "method")           # "A"
names(attributes(A))        # All available attributes

# --- Compact Mode (for large full-sib families) ---
A_compact <- pedmat(tped, method = "A", compact = TRUE)

# Query relationships (works for any individual, including merged sibs)
query_relationship(A_compact, "Z1", "Z2")  # Full-sibs Z1 and Z2

# View compression statistics
attr(A_compact, "compact_stats")
attr(A_compact, "family_summary")

# Expand back to full size
A_full <- expand_pedmat(A_compact)
dim(A_full)  # Original dimensions restored

# --- Inverse Matrices ---
Ainv <- pedmat(tped, method = "Ainv")  # Henderson's rules (fast)

# --- Dominance and Epistatic ---
D <- pedmat(tped, method = "D")
AA <- pedmat(tped, method = "AA")

# --- Visualization (requires display device) ---

vismat(A)                       # Heatmap of relationship matrix
vismat(A_compact)               # Works with compact matrices
vismat(A, by = "Gen")     # Group by generation
```

`pedmeta` *Access pedigree metadata from a tidyped object*

Description

Access pedigree metadata from a tidyped object

Usage

```
pedmeta(x)
```

Arguments

x	A tidyped object.

Value

The `ped_meta` list, or `NULL` if not set.

pedne *Calculate Effective Population Size*

Description

Calculates the effective population size (Ne) based on the rate of coancestry, the rate of inbreeding, or demographic parent numbers.

Usage

```
pedne(
  ped,
  method = c("coancestry", "inbreeding", "demographic"),
  by = NULL,
  reference = NULL,
  nsamples = 1000,
  ncores = 1,
  seed = NULL
)
```

Arguments

`ped`	A `tidyped` object.
`method`	Character. The method to compute Ne. One of `"coancestry"` (default), `"inbreeding"`, or `"demographic"`.
`by`	Character. The name of the column used to group cohorts (e.g., "Year", "BirthYear"). If NULL, calculates overall Ne for all individuals.
`reference`	Character vector. Optional subset of individual IDs defining the reference cohort. If NULL, uses all individuals in the pedigree.
`nsamples`	Integer. Number of individuals to randomly sample per cohort when using the `"coancestry"` method. Very large cohorts will be sampled down to this size to save memory and time (default: 1000).
`ncores`	Integer. Number of cores for parallel processing. Currently only effective for `method = "coancestry"` (default: 1).
`seed`	Integer or NULL. Random seed passed to `set.seed()` before sampling in the coancestry method, ensuring reproducible N_e estimates. Default is `NULL` (no fixed seed).

Details

The effective population size can be calculated using one of three methods:

- **"coancestry"** (Default): Based on the rate of coancestry (f_{ij}) between pairs of individuals. In this context, f_{ij} is the probability that two alleles randomly sampled from individuals i and j are identical by descent, which is equivalent to half the additive genetic relationship

($f_{ij} = a_{ij}/2$). This method is generally more robust as it accounts for full genetic drift and bottlenecks (Cervantes et al., 2011).

$$\Delta c_{ij} = 1 - (1 - c_{ij})^{1/(\frac{ECG_i + ECG_j}{2})}$$

$$N_e = \frac{1}{2\overline{\Delta c}}$$

To handle large populations, this method samples `nsamples` individuals per cohort and computes the mean rate of coancestry among them. Sampled relationship coefficients are obtained from batched matrix-free AX products, so the full additive relationship matrix is not constructed.

- **"inbreeding"**: Based on the individual rate of inbreeding (F_i) (Gutiérrez et al., 2008, 2009).

$$\Delta F_i = 1 - (1 - F_i)^{1/(ECG_i - 1)}$$

$$N_e = \frac{1}{2\overline{\Delta F}}$$

- **"demographic"**: Based on the demographic census of breeding males and females (Wright, 1931).

$$N_e = \frac{4N_m N_f}{N_m + N_f}$$

Where N_m and N_f are the number of unique male and female parents contributing to the cohort.

Value

A `data.table` with columns:

- `Cohort`: Cohort or grouping variable value.
- `N`: Number of individuals in the cohort.
- `Ne`: Calculated effective population size.
- `...`: Additional columns depending on the selected method (e.g., `NSampled`, `DeltaC`, `MeanF`, `DeltaF`, `Nm`, `Nf`).

References

Cervantes, I., Goyache, F., Molina, A., Valera, M., & Gutiérrez, J. P. (2011). Estimation of effective population size from the rate of coancestry in pedigreed populations. *Journal of Animal Breeding and Genetics*, 128(1), 56-63.

Gutiérrez, J. P., Cervantes, I., Molina, A., Valera, M., & Goyache, F. (2008). Individual increase in inbreeding allows estimating effective sizes from pedigrees. *Genetics Selection Evolution*, 40(4), 359-370.

Gutiérrez, J. P., Cervantes, I., & Goyache, F. (2009). Improving the estimation of realized effective population sizes in farm animals. *Journal of Animal Breeding and Genetics*, 126(4), 327-332.

Wright, S. (1931). Evolution in Mendelian populations. *Genetics*, 16(2), 97-159.

Examples

```
# Coancestry-based Ne (default) using a simple pedigree grouped by year
tp_simple <- tidyped(simple_ped)
tp_simple$BirthYear <- 2000 + tp_simple$Gen
ne_coan <- suppressMessages(pedne(tp_simple, by = "BirthYear", seed = 42L))
ne_coan

# Inbreeding-based Ne using an inbred pedigree
tp_inbred <- tidyped(inbred_ped)
ne_inb <- suppressMessages(pedne(tp_inbred, method = "inbreeding", by = "Gen"))
ne_inb

# Demographic Ne from the number of contributing sires and dams
ne_demo <- suppressMessages(pedne(tp_simple, method = "demographic", by = "BirthYear"))
ne_demo
```

`pedpartial` *Calculate Partial Inbreeding*

Description

Decomposes individuals' inbreeding coefficients into marginal contributions from specific ancestors. This allows identifying which ancestors or lineages are responsible for the observed inbreeding.

Usage

```
pedpartial(ped, ancestors = NULL, top = 20)
```

Arguments

`ped`	A `tidyped` object.
`ancestors`	Character vector. IDs of ancestors to calculate partial inbreeding for. If NULL, the top ancestors by marginal contribution are used.
`top`	Integer. Number of top ancestors to include if `ancestors` is NULL.

Details

The sum of all partial inbreeding coefficients for an individual (including contributions from founders) equals $1 + f_i$, where f_i is the total inbreeding coefficient. This function specifically isolates the terms in the Meuwissen & Luo (1992) decomposition that correspond to the selected ancestors.

Value

A `data.table` with the first column as `Ind` and subsequent columns representing the partial inbreeding (pF) from each ancestor.

References

Lacey, R. C. (1996). A formula for determining the partial inbreeding coefficient, F_{ij}. Journal of Heredity, 87(4), 337-339.

Meuwissen, T. H., & Luo, Z. (1992). Computing inbreeding coefficients in large populations. Genetics Selection Evolution, 24(4), 305-313.

Examples

```
library(data.table)
tp <- tidyped(inbred_ped)
# Calculate partial inbreeding originating from specific ancestors
target_ancestors <- inbred_ped[is.na(Sire) & is.na(Dam), Ind]
pF <- pedpartial(tp, ancestors = target_ancestors)
print(tail(pF))
```

`pedprod` *Matrix-Free Pedigree Relationship Products*

Description

Computes products of the form Ax, AX, $A^{-1}x$, or $A^{-1}X$ directly from an ordered pedigree, without constructing the full additive relationship matrix A. This is useful for large pedigrees where explicitly storing the dense $n \times n$ matrix is infeasible.

Usage

```
pedprod(ped, x, method = c("A", "Ainv"))
```

Arguments

`ped`	A complete pedigree accepted by `tidyped`, preferably an existing `tidyped` object.
`x`	A numeric or logical vector, or a numeric or logical matrix. An unnamed vector must have one value per pedigree individual in `IndNum` order. A named vector may contain a subset of individuals; omitted individuals receive value zero. For a matrix, the same rules apply to its rows and `rownames`.
`method`	Character scalar. `"A"` (default) computes products with the additive relationship matrix. `"Ainv"` computes products with its inverse.

Details

For an ordered pedigree, the additive relationship matrix can be written as

$$A = TDT',$$

where T is the transmission matrix and D contains Mendelian sampling variances. `pedprod()` applies these factors through backward and forward pedigree traversals. For an $n \times p$ right-hand side, computation requires $O(np)$ time and working memory rather than materializing the $O(n^2)$ relationship matrix.

Named inputs are aligned by individual ID. Every supplied name must occur in `ped$Ind`, and duplicate, missing, or empty names are rejected. Missing values and non-finite input values are also rejected to avoid propagating an invalid value through all descendants or relatives.

Value

For vector input, a named numeric vector in pedigree order. For matrix input, a numeric matrix whose rows are in pedigree order; input column names are preserved.

References

Colleau, J. J. (2002). An indirect approach to the extensive calculation of relationship coefficients. *Genetics Selection Evolution*, 34, 409–421. doi:10.1051/gse:2002015

Colleau, J. J., Palhiere, I., Rodriguez-Ramilo, S. T., & Legarra, A. (2017). A fast indirect method to compute functions of genomic relationships concerning genotyped and ungenotyped individuals, for diversity management. *Genetics Selection Evolution*, 49, 87. doi:10.1186/s1271101703639

See Also

`pedmat` for explicitly constructing relationship matrices and `pedrel` for group relationship summaries.

Examples

```
tped <- tidyped(small_ped)

# Equal contributions from two candidates; unspecified individuals are zero.
contributions <- c(Z1 = 0.5, Z2 = 0.5)
relationship_to_group <- pedprod(tped, contributions)

# Average additive relationship among the weighted candidates: c' A c.
sum(contributions * relationship_to_group[names(contributions)])

# Multiple contribution schemes can be evaluated in one call.
schemes <- cbind(
  Equal = c(A = 0.5, B = 0.5),
  SireA = c(A = 1.0, B = 0.0)
)
pedprod(tped, schemes)
```

`pedrel` *Calculate Mean Relationship or Coancestry Within Groups*

Description

Computes either the average pairwise additive genetic relationship coefficients (a_{ij}) within cohorts, or the corrected population mean coancestry used for pedigree-based diversity summaries.

Usage

```
pedrel(
  ped,
  by = "Gen",
  reference = NULL,
  compact = FALSE,
  scale = c("relationship", "coancestry")
)
```

Arguments

`ped`	A `tidyped` object.
`by`	Character. The column name to group by (e.g., "Year", "Breed", "Generation").
`reference`	Character vector. An optional vector of reference individual IDs to calculate relationships for. If provided, only individuals matching these IDs in each group will be used. Default is NULL (use all individuals in the group).
`compact`	Logical. Retained for backward compatibility. It is ignored because `pedrel()` now uses matrix-free pedigree products for all calculations.
`scale`	Character. One of `"relationship"` or `"coancestry"`. `"relationship"` returns the pairwise off-diagonal mean additive relationship (current `pedrel()` behavior). `"coancestry"` returns the corrected population mean coancestry used for pedigree-based diversity calculations.

Details

Let g be a zero-one indicator vector for the N selected individuals and let $S = g'Ag$. When `scale = "relationship"`, the returned value is

$$\bar{a}_{off} = \frac{S - \sum_{i:g_i=1}(1+F_i)}{N(N-1)},$$

the mean of the off-diagonal additive relationship coefficients. When `scale = "coancestry"`, the returned value is

$$\bar{C} = \frac{S}{2N^2},$$

which includes both pairwise coancestry and diagonal self-coancestry. This is equivalent to

$$\bar{C} = \frac{N-1}{N} \cdot \frac{\bar{a}_{off}}{2} + \frac{1+\bar{F}}{2N}.$$

The implementation computes Ag directly from the pedigree and does not construct the dense relationship matrix A.

Value

A `data.table` with columns:

- A grouping identifier column, named after the `by` parameter (e.g., `Gen`, `Year`).
- `NTotal`: Total number of individuals in the group.
- `NUsed`: Number of individuals used in calculation (could be subset by reference).
- `MeanRel`: Present when `scale = "relationship"`; average of off-diagonal elements in the Additive Relationship (A) matrix for this group ($a_{ij} = 2f_{ij}$).
- `MeanCoan`: Present when `scale = "coancestry"`; diagonal-corrected population mean coancestry for this group.
- `Status`: `"ok"`, `"skipped"`, or `"failed"`.
- `Message`: Empty for successful groups; otherwise a diagnostic explaining why `NA` was returned.

Examples

```
library(data.table)
# Use the sample dataset and simulate a birth year
tp <- tidyped(small_ped)
tp$Year <- 2010 + tp$Gen

# Example 1: Calculate average relationship grouped by Generation (default)
rel_by_gen <- pedrel(tp, by = "Gen")
print(rel_by_gen)

# Example 2: Calculate average relationship grouped by Year
rel_by_year <- pedrel(tp, by = "Year")
print(rel_by_year)

# Example 3: Calculate corrected mean coancestry
coan_by_gen <- pedrel(tp, by = "Gen", scale = "coancestry")
print(coan_by_gen)

# Example 4: Filter calculations with a reference list in a chosen group
candidates <- c("N", "O", "P", "Q", "T", "U", "V", "X", "Y")
rel_subset <- pedrel(tp, by = "Gen", reference = candidates)
print(rel_subset)
```

pedstats *Pedigree Statistics*

Description

Calculates comprehensive statistics for a pedigree, including population structure, generation intervals, and ancestral depth.

Usage

```
pedstats(
  ped,
  timevar = NULL,
  unit = "year",
  cycle = NULL,
  ecg = TRUE,
  genint = TRUE,
  ...
)
```

Arguments

ped	A tidyped object.
timevar	Optional character. Name of the column containing the **birth date** (or hatch date) of each individual. Accepted column formats: • Date or POSIXct (recommended). • A date string parseable by as.POSIXct (e.g., "2020-06-15"). Use format via ... for non-ISO strings. • A numeric year (e.g., 2020). Automatically converted to Date ("YYYY-07-01") with a message. If NULL, attempts auto-detection from common column names ("BirthYear", "Year", "BirthDate", etc.).
unit	Character. Time unit for reporting generation intervals: "year" (default), "month", "day", or "hour".
cycle	Numeric. Optional target generation cycle length in units. When provided, gen_intervals will include a GenEquiv column (observed Mean / cycle). See pedgenint for details.
ecg	Logical. Whether to compute equivalent complete generations for each individual via pedecg. Default TRUE.
genint	Logical. Whether to compute generation intervals via pedgenint. Requires a detectable timevar column. Default TRUE.
...	Additional arguments passed to pedgenint, e.g., format for custom date parsing or by for grouping.

Value

An object of class `pedstats`, which is a list containing:

- `summary`: A `data.table` with one row summarising the whole pedigree. Columns:
  - `N` — total number of individuals.
  - `NSire` — number of unique sires.
  - `NDam` — number of unique dams.
  - `NFounder` — number of founder individuals (both parents unknown).
  - `MaxGen` — maximum generation number.
- `ecg`: A `data.table` with one row per individual (`NULL` if `ecg = FALSE`). Columns:
  - `Ind` — individual identifier.
  - `ECG` — equivalent complete generations.
  - `FullGen` — number of fully known generations.
  - `MaxGen` — maximum traceable generation depth.
- `gen_intervals`: A `data.table` of generation intervals (`NULL` if no `timevar` is detected or `genint = FALSE`). Columns:
  - `Pathway` — gametic pathway label. Seven values: `"SS"` (sire to son), `"SD"` (sire to daughter), `"DS"` (dam to son), `"DD"` (dam to daughter) — require offspring sex; `"SO"` (sire to offspring) and `"DO"` (dam to offspring) — sex-independent; and `"Average"` — all parent-offspring pairs combined.
  - `N` — number of parent-offspring pairs.
  - `Mean` — mean generation interval.
  - `SD` — standard deviation of the interval.
  - `GenEquiv` — `Mean / cycle` (only present when `cycle` is supplied).

Examples

```
# ---- Without time variable ----
tp <- tidyped(simple_ped)
ps <- pedstats(tp)
ps$summary
ps$ecg

# ---- With annual Year column (big_family_size_ped) ----
tp2 <- tidyped(big_family_size_ped)
ps2 <- pedstats(tp2, timevar = "Year")
ps2$summary
ps2$gen_intervals
```

pedsubpop *Pedigree Subpopulations*

Description

Summarizes pedigree subpopulations and group structure.

Usage

```
pedsubpop(ped, by = NULL)
```

Arguments

ped	A tidyped object.
by	Character. The name of the column to group by. If NULL, summarizes disconnected components via splitped.

Details

When by = NULL, this function is a lightweight summary wrapper around splitped, returning one row per disconnected pedigree component plus an optional "Isolated" row for individuals with no known parents and no offspring. When by is provided, it instead summarizes the pedigree directly by the specified column (e.g. "Gen", "Year", "Breed").

Use pedsubpop() when you want a compact analytical summary table. Use splitped when you need the actual re-tidied sub-pedigree objects for downstream plotting or analysis.

Value

A data.table with columns:

- Group: Subpopulation label.
- N: Total individuals.
- N_Sire: Number of distinct sires.
- N_Dam: Number of distinct dams.
- N_Founder: Number of founders (parents unknown).

Examples

```
tp <- tidyped(simple_ped)

# Summarize disconnected pedigree components
pedsubpop(tp)

# Summarize by an existing grouping variable
pedsubpop(tp, by = "Gen")
```

plot.tidyped *Plot a tidy pedigree*

Description

Plot a tidy pedigree

Usage

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

Arguments

x	A tidyped object.
...	Additional arguments passed to visped.

Value

Invisibly returns a list of graph data from visped (node/edge data and layout components) used to render the pedigree; the primary result is the plot drawn on the current device.

print.pedcontrib *Print Founder and Ancestor Contributions*

Description

Print Founder and Ancestor Contributions

Usage

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

Arguments

x	A pedcontrib object.
...	Additional arguments.

print.pediv — *Print Genetic Diversity Summary*

Description

Print Genetic Diversity Summary

Usage

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

Arguments

x	A pediv object.
...	Additional arguments.

print.pedstats — *Print Pedigree Statistics*

Description

Print Pedigree Statistics

Usage

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

Arguments

x	A pedstats object.
...	Additional arguments.

print.summary.tidyped *Print method for summary.tidyped*

Description

Print method for summary.tidyped

Usage

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

Arguments

x	A summary.tidyped object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

print.tidyped *Print method for tidyped pedigree*

Description

Print method for tidyped pedigree

Usage

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

Arguments

x	A tidyped object
...	Additional arguments passed to the data.table print method

Value

The input object, invisibly.

query_relationship *Query Relationship Coefficients from a Pedigree Matrix*

Description

Retrieves relationship coefficients between individuals from a pedmat object. For compact matrices, automatically handles lookup of merged full-siblings.

Usage

```
query_relationship(x, id1, id2 = NULL)
```

Arguments

x	A pedmat object created by pedmat.
id1	Character, first individual ID.
id2	Character, second individual ID. If NULL, returns the entire row of relationships for id1.

Details

For compact matrices (compact = TRUE), this function automatically maps individuals to their family representatives. For methods A, D, and AA, it can compute the correct relationship even between merged full-siblings using the formula:

- **A**: $a_{ij} = 0.5 * (a_{is} + a_{id})$ where s, d are parents
- **D**: $d_{ij} = a_{ij}^2$ (for full-sibs in same family)
- **AA**: $aa_{ij} = a_{ij}^2$

Value

- If id2 is provided: numeric value (relationship coefficient)
- If id2 is NULL: named numeric vector (id1's row)
- Returns NA if individual not found

Note

Inverse matrices (Ainv, Dinv, AAinv) are **not supported** because inverse matrix elements do not represent meaningful relationship coefficients.

See Also

pedmat, expand_pedmat

Examples

```
tped <- tidyped(small_ped)
A <- pedmat(tped, method = "A", compact = TRUE)

# Query specific pair
query_relationship(A, "A", "B")

# Query merged full-siblings (works with compact)
query_relationship(A, "Z1", "Z2")

# Get all relationships for one individual
query_relationship(A, "A")
```

`simple_ped` *A simple pedigree*

Description

A small dataset containing a simple pedigree for demonstration.

Usage

```
simple_ped
```

Format

A data.table with 4 columns:

Ind Individual ID

Sire Sire ID

Dam Dam ID

Sex Sex of the individual

`small_ped` *A small pedigree*

Description

A small dataset containing a pedigree with some missing parents.

Usage

```
small_ped
```

Format

A data.frame with 3 columns:

Ind Individual ID

Sire Sire ID

Dam Dam ID

---

splitped — *Split Pedigree into Disconnected Groups*

Description

Detects and splits a tidyped object into disconnected groups (connected components). Uses igraph to efficiently find groups of individuals that have no genetic relationships with each other. Isolated individuals (Gen = 0, those with no parents and no offspring) are excluded from group splitting and stored separately.

Usage

```
splitped(ped)
```

Arguments

ped	A tidyped object created by `tidyped`.

Details

This function identifies connected components in the pedigree graph where edges represent parent-offspring relationships. Two individuals are in the same group if they share any ancestry (direct or indirect).

Isolated individuals (Gen = 0 in tidyped output) are those who:

- Have no known parents (Sire and Dam are both NA)
- Are not parents of any other individual in the pedigree

These isolated individuals are excluded from splitting and stored in the `isolated` attribute. Each resulting group contains at least 2 individuals (at least one parent-offspring relationship).

The function always returns a list, even if there is only one group (i.e., the pedigree is fully connected). Groups are sorted by size in descending order.

Each group in the result is a valid tidyped object with:

- Renumbered IndNum (1 to n for each group)
- Updated SireNum and DamNum referencing the new IndNum
- Recalculated Gen (generation) based on the group's structure

Value

A list of class "splitped" containing:

GP1, GP2, ...	tidyped objects for each disconnected group (with at least 2 individuals), with renumbered IndNum, SireNum, DamNum

The returned object has the following attributes:

n_groups	Number of disconnected groups found (excluding isolated individuals)
sizes	Named vector of group sizes
total	Total number of individuals in groups (excluding isolated)
isolated	Character vector of isolated individual IDs (Gen = 0)
n_isolated	Number of isolated individuals

See Also

tidyped for pedigree tidying

Examples

```
# Load example data
library(visPedigree)
data(small_ped)

# First tidy the pedigree
tped <- tidyped(small_ped)

# Split into groups
result <- splitped(tped)
print(result)

# Access individual groups (each is a tidyped object)
result$GP1

# Check isolated individuals
attr(result, "isolated")
```

---

summary.tidyped *Summary method for tidyped objects*

Description

Summary method for tidyped objects

Usage

```
## S3 method for class 'tidyped'
summary(object, ...)
```

Arguments

`object`	A tidyped object.
`...`	Additional arguments (ignored).

Value

A summary.tidyped object (list) containing:

- `n_ind`: Total number of individuals.
- `n_male`, `n_female`, `n_unknown_sex`: Sex composition counts.
- `n_founders`: Number of individuals with no known parents.
- `n_both_parents`: Count of individuals with complete parentage.
- `max_gen`, `gen_dist`: (Optional) Maximum generation and its distribution.
- `n_families`, `family_sizes`, `top_families`: (Optional) Family statistics.
- `f_stats`, `n_inbred`: (Optional) Inbreeding coefficient statistics.
- `n_cand`, `cand_f_stats`: (Optional) Candidate-specific statistics.

`summary_pedmat` *Summary Statistics for Pedigree Matrices*

Description

Computes and displays summary statistics for a pedmat object.

Usage

```
summary_pedmat(x)
```

Arguments

`x`	A pedmat object from `pedmat`.

Details

Since pedmat objects are often S4 sparse matrices with custom attributes, use this function instead of the generic `summary()` to ensure proper display of pedigree matrix statistics.

Value

An object of class `"summary.pedmat"` with statistics including method, dimensions, compression ratio (if compact), mean relationship, and matrix density.

See Also

`pedmat`

Examples

```
tped <- tidyped(small_ped)
A <- pedmat(tped, method = "A")
summary_pedmat(A)
```

`tidyped` *Tidy and prepare a pedigree*

Description

This function standardizes pedigree records, checks for duplicate IDs and incompatible parental roles, detects pedigree loops, injects missing founders, assigns generation numbers, sorts the pedigree, and optionally traces the pedigree of specified candidates. If the `cand` parameter contains individual IDs, only those individuals and their ancestors or descendants are retained. Tracing direction and the number of generations can be specified using the `trace` and `tracegen` parameters.

Usage

```
tidyped(
  ped,
  cand = NULL,
  trace = "up",
  tracegen = NULL,
  addgen = TRUE,
  addnum = TRUE,
  inbreed = FALSE,
  selfing = FALSE,
  genmethod = "top",
  ...
)
```

Arguments

`ped`	A data.table or data frame containing the pedigree. The first three columns must be **individual**, **sire**, and **dam** IDs. Additional columns, such as sex or generation, can be included. Column names can be customized, but their order must remain unchanged. Individual IDs should not be coded as "", " ", "0", "*", or "NA"; otherwise, they will be removed. Missing parents should be denoted by "NA", "0", or "*". Spaces and empty strings ("") are also treated as missing parents but are not recommended.
`cand`	A character vector of individual IDs, or NULL. If provided, only the candidates and their ancestors/descendants are retained.
`trace`	A character value specifying the tracing direction: "**up**", "**down**", or "**all**". "up" traces ancestors; "down" traces descendants; "all" traces the union of ancestors and descendants. This parameter is only used if `cand` is not NULL. Default is "up".

`tracegen` An integer specifying the number of generations to trace. This parameter is only used if `trace` is not NULL. If NULL or 0, all available generations are traced.

`addgen` A logical value indicating whether to generate generation numbers. Default is TRUE, which adds a **Gen** column to the output.

`addnum` A logical value indicating whether to generate a numeric pedigree. Default is TRUE, which adds **IndNum**, **SireNum**, and **DamNum** columns to the output.

`inbreed` A logical value indicating whether to calculate inbreeding coefficients. Default is FALSE. If TRUE, an **f** column is added to the output. This uses the same optimized engine as `pedmat(..., method = "f")`.

`selfing` A logical value indicating whether to allow the same individual to appear as both sire and dam. This is common in plant breeding (monoecious species) where the same plant can serve as both male and female parent. If TRUE, individuals appearing in both the Sire and Dam columns will be assigned Sex = "monoecious" instead of triggering an error. Default is FALSE.

`genmethod` A character value specifying the generation assignment method: "**top**" or "**bottom**". "top" (top-aligned) assigns generations from parents to offspring, starting founders at Gen 1. "bottom" (bottom-aligned) assigns generations from offspring to parents, aligning terminal nodes at the bottom. Default is "top".

`...` Additional arguments passed to `inbreed`.

Details

Compared to the legacy version, this function reports cyclic pedigrees more clearly and uses a mixed implementation. There are two candidate-tracing paths: when the input is a raw pedigree, `igraph` is used for loop checking, candidate tracing, and topological sorting; when the input is an already validated `tidyped` object and `cand` is supplied, tracing and topological sorting use integer-indexed C++ routines. Generation assignment can be performed using either a top-down approach (default, aligning founders at the top) or a bottom-up approach (aligning terminal nodes at the bottom).

Value

A `tidyped` object (which inherits from `data.table`). Individual, sire, and dam ID columns are renamed to **Ind**, **Sire**, and **Dam**. Missing parents are replaced with **NA**. The **Sex** column contains `"male"`, `"female"`, `"monoecious"`, or `NA`. The **Cand** column is included if `cand` is not NULL. The **Gen** column is included if `addgen` is TRUE. The **IndNum**, **SireNum**, and **DamNum** columns are included if `addnum` is TRUE. The **Family** and **FamilySize** columns identify full-sibling families (for example, `"AxB"` for offspring of sire `A` and dam `B`). The **f** column is included if `inbreed` is TRUE.

See Also

`summary.tidyped` for summarizing tidyped objects `visped` for visualizing pedigree structure `pedmat` for computing relationship matrices `vismat` for visualizing relationship matrices `splitped` for splitting pedigree into connected components `inbreed` for calculating inbreeding coefficients

Examples

```
library(visPedigree)
library(data.table)

# Tidy a simple pedigree
tidy_ped <- tidyped(simple_ped)
head(tidy_ped)

# Trace ancestors of a specific individual within 2 generations
tidy_ped_tracegen <- tidyped(simple_ped, cand = "J5X804", trace = "up", tracegen = 2)
head(tidy_ped_tracegen)

# Trace both ancestors and descendants for multiple candidates
# This is highly optimized and works quickly even on 100k+ individuals
cand_list <- c("J5X804", "J3Y620")
tidy_ped_all <- tidyped(simple_ped, cand = cand_list, trace = "all")

# Check for loops (will error if loops exist)
try(tidyped(loop_ped))

# Example with a large pedigree: extract 2 generations of ancestors for 2007 candidates
cand_2007 <- big_family_size_ped[Year == 2007, Ind]

tidy_big <- tidyped(big_family_size_ped, cand = cand_2007, trace = "up", tracegen = 2)
summary(tidy_big)
```

`vismat` *Visualize Relationship Matrices*

Description

`vismat` provides visualization tools for relationship matrices (A, D, AA), supporting individual-level heatmaps and relationship coefficient histograms. This function is useful for exploring population genetic structure, identifying inbred individuals, and analyzing kinship between families.

Usage

```
vismat(
  mat,
  ped = NULL,
  type = "heatmap",
  ids = NULL,
  reorder = TRUE,
  by = NULL,
  grouping = NULL,
  labelcex = NULL,
  ...
)
```

Arguments

`mat` A relationship matrix. Can be one of the following types:

- A `pedmat` object returned by `pedmat` — including compact matrices. When `by` is specified, group-level means are computed directly from the compact matrix (no full expansion needed). Without `by`, compact matrices are automatically expanded to full dimensions before plotting (see Details).
- A `tidyped` object (automatically calculates additive relationship matrix A).
- A named list containing matrices (preferring A, D, AA).
- A standard `matrix` or `Matrix` object.

Note: Inverse matrices (Ainv, Dinv, AAinv) are not supported for visualization because their elements do not represent meaningful relationship coefficients.

`ped` Optional. A tidied pedigree object (`tidyped`), used for extracting labels or grouping information. Required when using the `by` parameter with a plain matrix input. If `mat` is a `pedmat` object, the pedigree is extracted automatically.

`type` Character, type of visualization. Supported options:

- `"heatmap"`: Relationship matrix heatmap (default). Uses a Nature Genetics style color palette (white-orange-red-dark red), with optional hierarchical clustering and group aggregation.
- `"histogram"`: Distribution histogram of relationship coefficients. Shows the frequency distribution of lower triangular elements (pairwise kinship).

`ids` Character vector specifying individual IDs to display. Used to filter and display a submatrix of specific individuals. If `NULL` (default), all individuals are shown.

`reorder` Logical. If `TRUE` (default), rows and columns are reordered using hierarchical clustering (Ward.D2 method) to bring closely related individuals together. Only affects heatmap visualization. Automatically skipped for large matrices (N > VISMAT_REORDER_MAX, default 2 000) to improve performance.

Clustering principle: Based on relationship profile distance (Euclidean distance between rows). Full-sibs have nearly identical relationship profiles with the whole population, so they cluster tightly together and appear as contiguous blocks in the heatmap.

`by` Optional. Column name in `ped` to group by (e.g., `"Family"`, `"Gen"`, `"Year"`). When grouping is enabled:

- Individual-level matrix is aggregated to a group-level matrix (computing mean relationship coefficients between groups).
- For `"Family"` grouping, founders without family assignment are excluded.
- For other grouping columns, NA values are assigned to an `"Unknown"` group.

Useful for visualizing population structure in large pedigrees.

`grouping` `[Deprecated]` Use `by` instead.

`labelcex` Numeric. Manual control for font size of individual labels. If `NULL` (default), uses a dynamic font size that adjusts automatically based on matrix dimensions (range 0.2–0.7). Labels are hidden automatically when N > VISMAT_LABEL_MAX (default 500).

... Additional arguments passed to the underlying plotting function:

- Heatmap uses `levelplot`: can set `main`, `xlab`, `ylab`, `col.regions`, `colorkey`, `scales`, etc.
- Histogram uses `histogram`: can set `main`, `xlab`, `ylab`, `nint` (number of bins), etc.

Details

Compact Matrix Handling: When `mat` is a compact `pedmat` object (created with `pedmat(..., compact = TRUE)`):

- **With** `by`: Group-level mean relationships are computed algebraically from the K×K compact matrix, including a correction for sibling off-diagonal values. This avoids expanding to the full N×N matrix, making family-level or generation-level visualization feasible even for pedigrees with hundreds of thousands of individuals.
- **Without** `by`**, N > VISMAT_EXPAND_MAX (5 000)**: The compact K×K matrix is plotted directly using representative individuals. Labels show the number of individuals each representative stands for, e.g., `"ID (\u00d7350)"`. This avoids memory-intensive full expansion.
- **Without** `by`**, N ≤ 5 000**: The compact matrix is expanded via `expand_pedmat` to restore full dimensions.

Heatmap:

- Uses a Nature Genetics style color palette (white to orange to red to dark red).
- Hierarchical clustering reordering (Ward.D2) is enabled by default.
- Grid lines shown when N ≤ VISMAT_GRID_MAX (100); labels shown when N ≤ VISMAT_LABEL_MAX (500).
- `mat[1,1]` is displayed at the top-left corner.

Histogram:

- Shows the distribution of lower-triangular elements (pairwise kinship).
- X-axis: relationship coefficient values; Y-axis: frequency percentage.

Performance: The following automatic thresholds are defined as package-internal constants (`VISMAT_*`) at the top of `R/vismat.R`:

- Passing a `tidyped` object together with `by` uses matrix-free additive relationship products to compute the grouped heatmap directly, without constructing the full individual-level relationship matrix.
- `VISMAT_EXPAND_MAX` (5 000): compact matrices with original N above this are shown in representative view instead of expanding.
- `VISMAT_REORDER_MAX` (2 000): hierarchical clustering is automatically skipped.
- `VISMAT_LABEL_MAX` (500): individual labels are hidden.
- `VISMAT_GRID_MAX` (100): cell grid lines are hidden.
- `by` grouping uses vectorized `rowsum()` algebra — suitable for large matrices.

Interpreting Relationship Coefficients: For the additive relationship matrix A:

- Diagonal elements = 1 + F (F = inbreeding coefficient).
- Off-diagonal elements = 2 × kinship coefficient.
- 0: unrelated; 0.25: half-sibs / grandparent–grandchild; 0.5: full-sibs / parent–offspring; 1.0: same individual.

Value

Invisibly returns the `lattice` plot object. The plot is rendered on the current graphics device.

See Also

`pedmat` for computing relationship matrices, `expand_pedmat` for manually restoring compact matrix dimensions, `query_relationship` for querying individual pairs, `tidyped` for tidying pedigree data, `visped` for visualizing pedigree structure graphs, `levelplot`, `histogram`

Examples

```
library(visPedigree)
data(small_ped)
ped <- tidyped(small_ped)

# ============================================================
# Basic Usage
# ============================================================

# Method 1: from tidyped object (auto-computes A)
vismat(ped)

# Method 2: from pedmat object
A <- pedmat(ped)
vismat(A)

# Method 3: from plain matrix
vismat(as.matrix(A))

# ============================================================
# Compact Pedigree (auto-expanded before plotting)
# ============================================================

# For pedigrees with large full-sib families, compute a compact matrix
# first for efficiency, then pass directly to vismat() — it automatically
# expands back to full dimensions.
A_compact <- pedmat(ped, compact = TRUE)
vismat(A_compact)   # prints: "Expanding compact matrix (N -> M individuals)"

# For very large pedigrees, aggregate to a group-level view instead
vismat(A, ped = ped, by = "Gen",
       main = "Mean Relationship Between Generations")

# ============================================================
# Heatmap Customization
# ============================================================

# Custom title and axis labels
vismat(A, main = "Additive Relationship Matrix",
       xlab = "Individual", ylab = "Individual")

# Preserve original pedigree order (no clustering)
```

```
vismat(A, reorder = FALSE)

# Custom label font size
vismat(A, labelcex = 0.5)

# Custom color palette (blue-white-red)
vismat(A, col.regions = colorRampPalette(c("blue", "white", "red"))(100))

# ============================================================
# Display a Subset of Individuals
# ============================================================

target_ids <- rownames(A)[1:8]
vismat(A, ids = target_ids)

# ============================================================
# Histogram of Relationship Coefficients
# ============================================================

vismat(A, type = "histogram")
vismat(A, type = "histogram", nint = 30)

# ============================================================
# Group-level Aggregation
# ============================================================

# Group by generation
vismat(A, ped = ped, by = "Gen",
       main = "Mean Relationship Between Generations")

# Group by full-sib family (founders without a family are excluded)
vismat(A, ped = ped, by = "Family")

# ============================================================
# Other Relationship Matrices
# ============================================================

# Dominance relationship matrix
D <- pedmat(ped, method = "D")
vismat(D, main = "Dominance Relationship Matrix")
```

---

visped *Visualize a tidy pedigree*

Description

visped function draws a graph of a full or compact pedigree.

Usage

```
visped(
  ped,
  compact = FALSE,
  outline = FALSE,
  cex = NULL,
  showgraph = TRUE,
  file = NULL,
  highlight = NULL,
  trace = FALSE,
  showf = FALSE,
  labelvar = NULL,
  shapeby = c("sex", "role"),
  pagewidth = 200,
  symbolsize = 1,
  maxiter = 1000,
  genlab = FALSE,
  genlabcex = NULL,
  ...
)
```

Arguments

`ped` A `tidyped` object (which inherits from `data.table`). It is recommended that the pedigree is tidied and pruned by candidates using the `tidyped` function with the non-null parameter `cand`.

`compact` A logical value indicating whether terminal, non-parent full-sib individuals will be replaced by one family display node. For example, 100 eligible full siblings are shown as `"FS×100"` when `compact = TRUE`. Highlighted individuals are not compacted, and the underlying pedigree is unchanged. The default value is FALSE.

`outline` A logical value indicating whether shapes without labels will be shown. A graph of the pedigree without individual labels is shown when setting `outline = TRUE`. This is useful for viewing the pedigree outline and identifying immigrant individuals in each generation when the graph width exceeds the maximum PDF width (500 inches). The default value is FALSE.

`cex` NULL or a numeric value changing the size of individual labels shown in the graph. *cex* is an abbreviation for 'character expansion factor'. The `visped` function will attempt to estimate (`cex=NULL`) the appropriate cex value and report it in the messages. Based on the reported cex from a previous run, this parameter should be increased if labels are wider than their shapes in the PDF; conversely, it should be decreased if labels are narrower than their shapes. The default value is NULL.

`showgraph` A logical value indicating whether a plot will be shown in the default graphic device (e.g., the Plots panel in RStudio). This is useful for quick viewing without opening a PDF file. However, the graph on the default device may not be legible (e.g., overlapping labels or aliasing lines) due to size restrictions. It is

recommended to set `showgraph = FALSE` for large pedigrees. The default value is TRUE.

`file` NULL or a character value specifying whether the pedigree graph will be saved as a vector file. Files ending in '.svg' are written with the SVG device; all other file names use PDF output. The vector output is legible and avoids overlapping labels even with many individuals or long labels. It is recommended to save the pedigree graph as a vector file. The default value is NULL.

`highlight` NULL, a character vector of individual IDs, or a list specifying individuals to highlight. If a character vector is provided, individuals will be highlighted with a purple border while preserving their sex-based fill color. If a list is provided, it should contain:

- `ids`: (required) character vector of individual IDs to highlight.
- `frame.color`: (optional) hex color for the border of focal individuals.
- `color`: (optional) hex color for the fill of focal individuals.
- `rel.frame.color`: (optional) hex color for the border of relatives (used when `trace` is not NULL).
- `rel.color`: (optional) hex color for the fill of relatives (used when `trace` is not NULL).

For example: `c("A", "B")` or `list(ids = c("A", "B"), frame.color = "#9c27b0")`. The function will check if the specified individuals exist in the pedigree and issue a warning for any missing IDs. The default value is NULL.

`trace` A logical value or a character string. If TRUE, all ancestors and descendants of the individuals specified in `highlight` will be highlighted. If a character string, it specifies the tracing direction: "**up**" (ancestors), "**down**" (descendants), or "**all**" (union of ancestors and descendants). This is useful for focusing on specific families within a large pedigree. The default value is FALSE.

`showf` A logical value indicating whether inbreeding coefficients will be shown in the graph. If `showf = TRUE` and the column **f** is missing, `visped()` will try to compute it automatically with `inbreed` on a structurally complete pedigree. If automatic computation is not possible, a warning is issued and labels are drawn without **f**. The default value is FALSE.

`labelvar` NULL, a single non-empty character value naming a column in `ped`, or a character vector with one displayed label per input pedigree row. Row-aligned vectors are attached before any internal tidying or reordering. Compact full-sib family nodes still show family size when `compact = TRUE`. Missing or empty values fall back to individual IDs. The default value is NULL.

`shapeby` Character string controlling what node shapes encode. `"sex"` (default) uses circles for females, squares for males, diamonds for unknown sex, and hexagons for monoecious individuals. `"role"` distinguishes individual records from compact family summaries by shape while keeping sex-based fill colors for real individuals. Compact family summaries are rectangles in both modes.

`pagewidth` A numeric value specifying the width of the PDF file in inches. This controls the horizontal scaling of the layout. The default value is 200.

`symbolsize` A numeric value specifying the scaling factor for node size relative to the label size. Values greater than 1 increase the node size (adding padding around the

label), while values less than 1 decrease it. This is useful for fine-tuning the whitespace and legibility of dense graphs. The default value is 1.

`maxiter` An integer specifying the maximum number of iterations for the Sugiyama layout algorithm to minimize edge crossings. Higher values (e.g., 2000 or 5000) may result in fewer crossed lines for complex pedigrees but will increase computation time. The default value is 1000.

`genlab` FALSE to omit generation labels; TRUE to draw the default labels G1, G2, ...; or an unnamed, non-empty character vector of custom labels. Custom labels are assigned from top to bottom to the final displayed generation layers and must contain exactly one label per layer. The default value is FALSE.

`genlabcex` NULL or a numeric value controlling the size of displayed generation labels. If `NULL`, `visped()` uses an automatic size based on node scaling. Set a larger value to keep generation labels readable in deep pedigrees. The default value is NULL.

`...` Additional arguments passed to `plot.igraph`.

Details

This function takes a pedigree tidied by the `tidyped` function and outputs a hierarchical graph for all individuals in the pedigree. The graph can be shown on the default graphic device or saved as a PDF or SVG file. The vector output is legible and avoids overlapping labels. It is especially useful when the number of individuals is large and individual labels are long.

Rendering is performed using a Two-Pass strategy: edges are drawn first to ensure center-to-center connectivity, followed by nodes and labels. This ensures perfect visual alignment in high-resolution vector outputs. The function also supports real-time ancestry and descendant highlighting.

This function can draw the graph of a very large pedigree (> 10,000 individuals per generation) by compacting full-sib individuals. It is highly effective for aquatic animal pedigrees, which usually include many full-sib families per generation in nucleus breeding populations. The outline of a pedigree without individual labels is still shown if the width of a pedigree graph exceeds the maximum width (500 inches) of the PDF file.

By default, `shapeby = "sex"` encodes individual sex with node shape: females are circles, males are squares, individuals of unknown sex are diamonds, and monoecious individuals are hexagons. Set `shapeby = "role"` to use the legacy role-based scheme, where circles represent individual records and green-grey rectangles represent compact full-sib family summaries. Compact family summaries remain green-grey rectangles in both modes. In `shapeby = "role"` mode, real individuals remain circles, but their fill colors still encode sex. Dark sky blue indicates males, dark goldenrod indicates females, teal indicates monoecious individuals, and neutral grey indicates unknown sex. Purple borders indicate highlighted individuals rather than sex.

Value

The function mainly produces a plot on the current graphics device and/or a vector file. It invisibly returns a list containing the graph object, layout coordinates, and node sizes.

Note

Isolated individuals (those with no parents and no progeny, assigned Gen 0) are automatically filtered out and not shown in the plot. A message will be issued if any such individuals are removed.

See Also

tidyped for tidying pedigree data (required input) vismat for visualizing relationship matrices as heatmaps pedmat for computing relationship matrices splitped for splitting pedigree into connected components plot.igraph underlying plotting function

Examples

```
library(visPedigree)
library(data.table)
# Drawing a simple pedigree
simple_ped_tidy <- tidyped(simple_ped)
visped(simple_ped_tidy,
       cex=0.25,
       symbolsize=5.5)

# Use application-specific labels for the displayed generations.
# Custom labels are assigned from top to bottom.
generation_labels <- paste("Generation", sort(unique(simple_ped_tidy$Gen)))
visped(simple_ped_tidy,
       genlab = generation_labels,
       cex = 0.25,
       symbolsize = 5.5)

# Highlighting an individual and its ancestors and descendants
visped(simple_ped_tidy,
       highlight = "J5X804",
       trace = "all",
       cex=0.25,
       symbolsize=5.5)

# Showing inbreeding coefficients in the graph
simple_ped_tidy_inbreed <- tidyped(simple_ped, inbreed = TRUE)
visped(simple_ped_tidy_inbreed,
       showf = TRUE,
       cex=0.25,
       symbolsize=5.5)

# visped() will automatically compute inbreeding coefficients if 'f' is missing
visped(simple_ped_tidy,
       showf = TRUE,
       cex=0.25,
       symbolsize=5.5)

# Use the role-based legacy symbol scheme
visped(simple_ped_tidy,
       shapeby = "role",
       cex=0.25,
       symbolsize=5.5)

# Adjusting page width and symbol size for better layout
# Increase pagewidth to spread nodes horizontally in the pdf file
# Increase symbolsize for more padding around individual labels
```

```
visped(simple_ped_tidy,
       cex=0.25,
       symbolsize=5.5,
       pagewidth = 100,
       file = tempfile(fileext = ".pdf"))

# Saving pedigree as SVG with larger labels and tighter symbols
visped(simple_ped_tidy,
       cex=0.8,
       symbolsize=1,
       pagewidth = 100,
       file = tempfile(fileext = ".svg"))

# Highlighting multiple individuals with custom colors
visped(simple_ped_tidy,
       highlight = list(ids = c("J3Y620", "J1X971"),
                        frame.color = "#4caf50",
                        color = "#81c784"),
       cex=0.25,
       symbolsize=5.5)

# Handling large pedigrees: Saving to PDF is recommended for legibility
# The 'trace' and 'tracegen' parameters in tidyped() help prune the graph
cand_labels <- big_family_size_ped[(Year == 2007) & (substr(Ind,1,2) == "G8"), Ind]

big_ped_tidy <- tidyped(big_family_size_ped,
                        cand = cand_labels,
                        trace = "up",
                        tracegen = 2)
# Use compact = TRUE for large families
visped(big_ped_tidy,
       compact = TRUE,
       cex=0.08,
       symbolsize=5.5,
       file = tempfile(fileext = ".pdf"))

# Use outline = TRUE if individual labels are not required
visped(big_ped_tidy,
       compact = TRUE,
       outline = TRUE,
       file = tempfile(fileext = ".pdf"))
```

`[.tidyped` *Subset a tidyped object*

Description

Intercepts `data.table`'s `[` method for `tidyped` objects. After subsetting, the method checks whether the result is still a valid pedigree (all referenced parents still present). If so, `IndNum`,

SireNum, and DamNum are rebuilt and the tidyped class is preserved. If the pedigree becomes structurally incomplete (missing parent records), the result is degraded to a plain data.table with a warning. Column-only selections (missing core columns) also return a plain data.table.

Usage

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

Arguments

x	A tidyped object.
...	Arguments passed to the data.table [method.

Value

A tidyped object if the result is still a complete pedigree, otherwise a plain data.table.

Index

∗ **datasets**
big_family_size_ped, 4
complex_ped, 5
deep_ped, 5
half_founder_ped, 7
inbred_ped, 8
loop_ped, 10
simple_ped, 40
small_ped, 40
[.tidyped, 55

as_tidyped, 3

big_family_size_ped, 4

complex_ped, 5

deep_ped, 5

expand_pedmat, 6, *23, 24*, *39*, *48, 49*

half_founder_ped, 7
has_candidates, 7
has_inbreeding, 8
histogram, *48, 49*

inbred_ped, 8
inbreed, 9, *24*, *45*, *52*
is_tidyped, 10

levelplot, *48, 49*
loop_ped, 10

new_tidyped, *4*

pedancestry, 11
pedcontrib, 12, *18, 19*, *21*
pedecg, 14, *33*
pedfclass, 15
pedgenint, 16, *33*
pedhalflife, 18
pediv, *19*, 20
pedmat, *6*, 22, *30*, *39*, *43*, *45*, *47*, *49*, *54*
pedmeta, 25
pedne, *4*, *21*, 26
pedpartial, 28
pedprod, *24*, 29
pedrel, *30*, 31
pedstats, *4*, *21*, 33
pedsubpop, 35
plot.igraph, *53, 54*
plot.pedhalflife *(*pedhalflife*)*, 18
plot.tidyped, 36
print.pedcontrib, 36
print.pedhalflife *(*pedhalflife*)*, 18
print.pediv, 37
print.pedstats, 37
print.summary.tidyped, 38
print.tidyped, 38

query_relationship, *6*, *23, 24*, 39, *49*

simple_ped, 40
small_ped, 40
splitped, *35*, 41, *45*, *54*
summary.tidyped, 42, *45*
summary_pedmat, *23*, 43

tidyped, *3, 4*, *9*, *19*, *22*, *24*, *29*, *41, 42*, 44, *47*, *49*, *51*, *53, 54*

vismat, *23, 24*, *45*, 46, *54*
visped, *36*, *45*, *49*, 50