Package 'nprcgenekeepr'

November 2, 2026

Type Package

Title Genetic Tools for Colony Management

Version 3.0.0

Description Provides genetic tools for colony management and implements a derivation of the work in Amanda Vinson and Michael J Raboin (2015) <https://pmc.ncbi.nlm.nih.gov/articles/PMC4671785/> ``A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques ('Macaca' 'mulatta')''.
It provides a 'Shiny' application with an exposed API.
The application supports these main groups of functions:
(1) Quality control of studbooks contained in text files or 'Excel' workbooks and of pedigrees within 'LabKey' Electronic Health Records (EHR);
(2) Creation of pedigrees from a list of animals using the 'LabKey' EHR integration;
(3) Creation and display of an age by sex pyramid plot of the living animals within the designated pedigree;
(4) Generation of genetic value analysis reports; and
(5) Creation of potential breeding groups with and without proscribed sex ratios and defined maximum kinships.
Further tabs cover mate pair analysis, genetic diversity, marker genetics, potential parents, cross-center identity mapping, de-identified export, and genetic-health trends.

URL https://rmsharp.github.io/nprcgenekeepr/, https://github.com/rmsharp/nprcgenekeepr

BugReports https://github.com/rmsharp/nprcgenekeepr/issues

Depends R (>= 4.1.0)

Imports anytime, bslib, data.table, DT, futile.logger, ggplot2, htmlTable, lifecycle, lubridate, Matrix, openxlsx, plotrix, quadprog, readxl, Rlabkey (>= 3.2.0), sessioninfo, shiny, shinyBS, stringi, utils, visNetwork

Suggests chromote, dplyr, shinytest2, htmltools, htmlwidgets, kableExtra, kinship2, knitr, markdown, mockery, pkgdown, rmarkdown, spelling, testthat, withr, yaml

Language en-US

Encoding UTF-8

License MIT + file LICENSE

LazyData TRUE

VignetteBuilder knitr

Config/renv/profiles/dev/dependencies checkhelper, devtools, httpuv, qpdf, markdown, pak, roxygen2, spelling, usethis

Config/Needs/website quarto

Config/Needs/coverage covr

Config/Needs/dev devtools, roxygen2

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Michael Raboin [aut],
Terry Therneau [aut],
Amanda Vinson [aut, dtc],
R. Mark Sharp [aut, cre, cph, dtc] (ORCID: <https://orcid.org/0000-0002-6170-6942>),
Matthew Schultz [aut] (ORCID: <https://orcid.org/0000-0001-5103-4305>),
Southwest National Primate Research Center [fnd] (NIH grant P51 RR13986, work up to and including version 1.0),
Oregon National Primate Research Center [fnd] (NIH grant P51 OD011092, work up to and including version 1.0)

Maintainer R. Mark Sharp <rmsharp@me.com>

Repository CRAN

Date/Publication 2026-11-01 23:40:02 UTC

Contents

addAnimalsWithNoRelative 8
addBackSecondParents 9
addGenotype 10
addIdRecords 11
addParents 12
addSexAndAgeToGroup 13
addUIds 13
alleleFreq 14
appendColonySnapshot 15
applyKinshipOverrides 16
appServer 17
appUI 18
assignAlleles 19
backfillColonySnapshots 20
buildMarkerGenotypeMatrix 22

calcA . . . 24
calcAge . . . 25
calcFE . . . 25
calcFEFG . . . 27
calcFG . . . 28
calcFGSE . . . 30
calcGeneDiversity . . . 31
calcGU . . . 32
calcGUSE . . . 34
calcKurtosis . . . 35
calcNeSexRatio . . . 36
calcNeVariance . . . 37
calcRetention . . . 39
calcSkewness . . . 40
calcSnapshotDeltas . . . 41
calculateSexRatio . . . 42
checkAncestryRules . . . 43
checkChangedColsLst . . . 44
checkCrossCenterMapping . . . 45
checkErrorLst . . . 47
checkGenotypeFile . . . 47
checkKinshipOverrides . . . 48
checkLinkageMarkerGenotypeFile . . . 49
checkLocusMetadata . . . 50
checkMarkerGenotypeFile . . . 51
checkMhcHaplotypeFile . . . 53
checkParentAge . . . 54
checkRequiredCols . . . 55
checkSequenceGenotypeFile . . . 56
checkSnapshotHistory . . . 58
checkTwinRelations . . . 59
chooseAlleles . . . 60
chooseDate . . . 61
computeGenomicROH . . . 62
convertAncestry . . . 64
convertDate . . . 64
convertFromCenter . . . 66
convertRelationships . . . 67
convertSexCodes . . . 68
convertStatusCodes . . . 69
correctParentSex . . . 70
countFirstOrder . . . 72
countKinshipValues . . . 73
countLoops . . . 74
createColonySnapshot . . . 75
createExampleFiles . . . 77
createPedTree . . . 77
createSimKinships . . . 78

create_wkbk . . . 79
cumulateSimKinships . . . 80
dataframe2string . . . 82
drawPedigreeDiagram . . . 83
exampleNprcgenekeeprConfig . . . 84
examplePedigree . . . 84
fillGroupMembersWithSexRatio . . . 85
filterKinMatrix . . . 87
filterPairs . . . 88
filterReport . . . 89
filterThreshold . . . 90
finalRpt . . . 91
findGeneration . . . 91
findLoops . . . 92
findOffspring . . . 93
findPedigreeNumber . . . 94
fixColumnNames . . . 95
focalAnimals . . . 95
geneDrop . . . 96
getAncestors . . . 98
getAnimalsWithHighKinship . . . 99
getAutoIdFormat . . . 100
getBoxWhiskerDescription . . . 101
getChangedColsTab . . . 102
getConfigFileName . . . 102
getCurrentAge . . . 103
getDatedFilename . . . 103
getDateErrorsAndConvertDatesInPed . . . 104
getDemographics . . . 105
getDescendantPedigree . . . 106
getEmptyErrorLst . . . 106
getErrorTab . . . 107
getFileDirectRelatives . . . 107
getFocalAnimalPed . . . 109
getFocalAnimalPedFromFile . . . 110
getFounders . . . 111
getGeneticDiversityStats . . . 112
getGenotypes . . . 114
getGVGenotype . . . 114
getGVPopulation . . . 115
getIdsWithOneParent . . . 116
getIncludeColumns . . . 117
getLkDirectAncestors . . . 118
getLkDirectRelatives . . . 118
getOffspring . . . 119
getParents . . . 120
getPedDirectRelatives . . . 121
getPedigree . . . 122

getPedMaxAge . . . 122
getPossibleCols . . . 123
getPotentialParents . . . 125
getPotentialSires . . . 126
getProbandPedigree . . . 127
getPyramidAgeDist . . . 128
getPyramidPlot . . . 129
getRequiredCols . . . 130
getSiteInfo . . . 130
getSpeciesGestation . . . 131
getSpeciesMinBreedingAge . . . 132
getTokenList . . . 133
getVersion . . . 134
get_and_or_list . . . 135
get_elapsed_time_str . . . 135
groupAddAssign . . . 136
gvaConvergence . . . 140
hasBothParents . . . 143
hasGenotype . . . 143
headerDisplayNames . . . 144
isFounder . . . 145
is_valid_date_str . . . 146
kinMatrix2LongForm . . . 146
kinship . . . 147
kinshipMatricesToKValues . . . 149
kinshipMatrixToKValues . . . 151
lacy1989Ped . . . 153
lacy1989PedAlleles . . . 154
loadSiteConfig . . . 154
loadSpeciesOverrides . . . 155
logModuleEvent . . . 156
makeCEPH . . . 157
makeExamplePedigreeFile . . . 158
makeFounderStatsTable . . . 159
makeGeneticDiversityHeatmap . . . 160
makeGeneticSummaryTable . . . 161
makeGroupMembers . . . 162
makeGroupNum . . . 163
makeGrpNum . . . 163
makePedigreeDiagramData . . . 164
makePedigreeMatingLayout . . . 165
makeRelationClassesTable . . . 167
makeSimPed . . . 168
mapIdsToObfuscated . . . 169
markerExpectedHeterozygosity . . . 170
markerFst . . . 171
markerKinship . . . 173
markerLdBlock . . . 175

markerObservedHeterozygosity . . . 176
markerParentageExclusion . . . 177
markerParentageLikelihood . . . 179
markerRealizedRelatednessVariance . . . 182
meanKinship . . . 184
mhcHaplotypeCarriers . . . 185
mhcHaplotypeFrequency . . . 186
modBreedingGroupsServer . . . 188
modBreedingGroupsUI . . . 191
modCrossCenterIdentityServer . . . 192
modCrossCenterIdentityUI . . . 193
modDeidentifiedExportServer . . . 194
modDeidentifiedExportUI . . . 195
modGeneticDiversityServer . . . 196
modGeneticDiversityUI . . . 197
modGeneticValueServer . . . 198
modGeneticValueUI . . . 199
modGvAndBgDescServer . . . 200
modGvAndBgDescUI . . . 201
modInputServer . . . 202
modInputUI . . . 203
modMarkerGeneticsServer . . . 204
modMarkerGeneticsUI . . . 206
modMatePairServer . . . 207
modMatePairUI . . . 210
modORIPReportingServer . . . 210
modORIPReportingUI . . . 211
modPedigreeServer . . . 212
modPedigreeUI . . . 214
modPotentialParentsServer . . . 215
modPotentialParentsUI . . . 216
modPyramidServer . . . 217
modPyramidUI . . . 218
modSnapshotTrendsServer . . . 218
modSnapshotTrendsUI . . . 220
modSummaryStatsServer . . . 221
modSummaryStatsUI . . . 223
obfuscateDate . . . 224
obfuscateGenomicROH . . . 225
obfuscateGenotypeMatrix . . . 226
obfuscateId . . . 227
obfuscateLdBlocks . . . 228
obfuscateMhcHaplotypes . . . 229
obfuscatePed . . . 230
obfuscateTwinRelations . . . 232
offspringCounts . . . 233
ped1Alleles . . . 234
pedDuplicateIds . . . 235

pedFemaleSireMaleDam . . . 235
pedGood . . . 236
pedInvalidDates . . . 236
pedMissingBirth . . . 237
pedOne . . . 237
pedSameMaleIsSireAndDam . . . 238
pedSix . . . 238
pedWithGenotype . . . 239
pedWithGenotypeReport . . . 239
plotSnapshotTrends . . . 240
previewBackfillSupport . . . 241
print.summary.nprcgenekeeprErr . . . 242
processQcStudbookResult . . . 243
qcBreeders . . . 244
qcPed . . . 245
qcPedGvReport . . . 246
qcStudbook . . . 246
rankSubjects . . . 250
readAncestryRules . . . 251
readKinshipOverrides . . . 252
readSnapshotHistory . . . 253
readTwinRelations . . . 254
removeAutoGenIds . . . 255
removeDuplicates . . . 255
removeEarlyDates . . . 256
removePotentialSires . . . 257
removeUninformativeFounders . . . 258
removeUnknownAnimals . . . 259
reportAncestryViolations . . . 260
reportGV . . . 261
reportMatePairs . . . 264
resolveCrossCenterIds . . . 268
rhesusGenotypes . . . 269
rhesusPedigree . . . 270
runGeneKeepR . . . 271
runModularApp . . . 272
runQcStudbook . . . 273
safeExecute . . . 274
saveDataframesAsFiles . . . 275
savePlotToFile . . . 276
setAutoIdFormat . . . 277
setExit . . . 278
setLabKeyDefaults . . . 279
setPopulation . . . 280
set_seed . . . 281
shouldShowChangedColsTab . . . 282
shouldShowOripTab . . . 282
shrinkPedigree . . . 283

smallPed 285
smallPedTree 286
speciesGestation 286
summarizeKinshipValues 287
summary.nprcgenekeeprErr 289
toCharacter 290
trimPedigree 291
withinIntegerRange 292

Index **294**

`addAnimalsWithNoRelative` *Add an NA value for animals with no relative*

Description

This allows `kin` to be used with `setdiff` when there are no relatives otherwise an error would occur because `kin[['animal_with_no_relative']]` would not be found. See the following: in **groupAddAssign**

Usage

```
addAnimalsWithNoRelative(kin, candidates)
```

Arguments

`kin`	named list of high-kinship relatives, as produced by `getAnimalsWithHighKinship`, where each name is an animal Id and each value is a character vector of the Ids sharing a kinship value at or above the threshold.
`candidates`	character vector of IDs of the animals available for use in the group.

Details

```
available[[i]] <- setdiff(available[[i]], kin[[id]])
```

Value

The named list of high-kinship relatives (one element per animal Id, each value a character vector of that Id's high-kinship relatives) with an added element set to `NA` for each candidate that has no relative.

Examples

```
library(nprcgenekeepr)
qcPed <- nprcgenekeepr::qcPed
ped <- qcStudbook(qcPed,
  minSireAge = 2, minDamAge = 2, reportChanges = FALSE,
  reportErrors = FALSE
)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
currentGroups <- list(1L)
currentGroups[[1]] <- examplePedigree$id[1:3]
candidates <- examplePedigree$id[examplePedigree$status == "ALIVE"]
threshold <- 0.015625
kin <- getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups,
  ignore = list(c("F", "F")), minAge = 1.0
)
# Filtering out candidates related to current group members
conflicts <- unique(c(
  unlist(kin[unlist(currentGroups)]),
  unlist(currentGroups)
))
candidates <- setdiff(candidates, conflicts)
kin <- addAnimalsWithNoRelative(kin, candidates)
length(kin) # should be 591
kin[["0DAV0I"]] # should have 34 IDs
```

---

addBackSecondParents *Add back second parents to a trimmed pedigree*

Description

Uses the ped dataframe, which has full complement of parents and the uPed dataframe, which has all uninformative parents removed to add back single parents to the uPed dataframe where one parent is known. The parents are added back to the pedigree as an ID record with NA for both sire and dam of the added back ID.

Usage

```
addBackSecondParents(uPed, ped)
```

Arguments

uPed	a trimmed pedigree dataframe with uninformative founders removed.
ped	the full (untrimmed) pedigree dataframe, from which the second parents are taken.

Value

A pedigree with single parents added. It is a `data.table` once any record is added and a plain dataframe when none is.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2, minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
probands <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
p <- addBackSecondParents(p, ped)
nrow(p)
```

---

addGenotype *Add genotype data to pedigree file*

Description

Assumes genotype has been opened by `checkGenotypeFile`

Usage

```
addGenotype(ped, genotype)
```

Arguments

`ped`	pedigree dataframe. `ped` is to be provided by `qcStudbook` so it is not checked.
`genotype`	genotype dataframe. `genotype` is to be provided by `checkGenotypeFile` so it is not checked.

Details

The two allele columns are coerced to character internally so the name-keyed allele dictionary is both built and indexed by allele label. This keeps the integer encoding consistent even when the allele columns are supplied as factors (a factor would otherwise be indexed by its integer codes). The allele columns are taken by position (columns 2 and 3 of `genotype`), not by name. The result is the output of a full outer `merge()` on `id`, so genotype ids that are absent from `ped` are added as extra rows.

Value

A plain `data.frame` (not a special pedigree class) with the rows of `ped` plus integer columns `first` and `second` holding the allele codes (numbered from 10001).

Examples

```
library(nprcgenekeepr)
rhesusPedigree <- nprcgenekeepr::rhesusPedigree
rhesusGenotypes <- nprcgenekeepr::rhesusGenotypes
pedWithGenotypes <- addGenotype(
  ped = rhesusPedigree,
  genotype = rhesusGenotypes
)
```

`addIdRecords` *Add ego records with NA parent IDs*

Description

Add ego records with NA parent IDs

Usage

```
addIdRecords(ids, fullPed, partialPed)
```

Arguments

`ids`	character vector of IDs to be added as Ego records having NAs for parent IDs
`fullPed`	the full (untrimmed) pedigree dataframe, the source of the records to add. It must have the same columns as `partialPed`.
`partialPed`	a trimmed pedigree dataframe with uninformative founders removed.

Value

Pedigree with Ego records added having NAs for parent IDs. It is a `data.table` once any record is added and a plain dataframe when none is.

Examples

```
uPedOne <- data.frame(
  id = c("d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c("s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c("d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("F", "M", "F", "F", "F", "F", "M"),
  stringsAsFactors = FALSE
)
pedOne <- data.frame(
  id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
  stringsAsFactors = FALSE
```

```
)
pedOne[!pedOne$id %in% uPedOne$id, ]
newPed <- addIdRecords(ids = "s1", pedOne, uPedOne)
pedOne[!pedOne$id %in% newPed$id, ]
newPed[newPed$id == "s1", ]
```

---

addParents *Add parents*

Description

Pedigree curation function Given a pedigree, find any IDs listed in the "sire" or "dam" columns that lack their own line entry and generate one.

Usage

```
addParents(ped)
```

Arguments

ped	dataframe that is the Pedigree. The id, sire, dam and sex columns are required.

Details

This must be run after addUIds since the IDs made there are used by addParents

Value

An updated pedigree with entries added as necessary. Entries have the id and sex specified; all remaining columns except recordStatus are filled with NA. A recordStatus column is added (replacing any existing one) with "original" for the existing records and "added" for the new ones.

Examples

```
pedTwo <- data.frame(
  id = c("d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, NA, NA, "s1", "s1", "s2", "s2"),
  dam = c(NA, NA, NA, "d1", "d2", "d2", "d2"),
  sex = c("F", "M", "F", "F", "F", "F", "M"),
  stringsAsFactors = FALSE
)
newPed <- addParents(pedTwo)
newPed
```

addSexAndAgeToGroup *Build a group data frame with ID, sex, and age*

Description

Build a group data frame with ID, sex, and age

Usage

```
addSexAndAgeToGroup(ids, ped)
```

Arguments

ids	character vector of animal IDs
ped	The pedigree information in data.frame format

Details

An empty ids vector yields a zero-row data frame that still contains all three columns (ids, sex, age), with sex an empty factor, so the returned schema does not depend on the number of ids supplied.

Value

A data frame with columns ids, sex and age (current age). Every id must occur exactly once in ped$id; an id that is missing from or duplicated in ped causes an error rather than an NA.

Examples

```
library(nprcgenekeepr)
data("qcBreeders")
data("qcPed")
df <- addSexAndAgeToGroup(ids = qcBreeders, ped = qcPed)
head(df)
```

addUIds *Add placeholder IDs for unknown parents*

Description

This must be run prior to addParents since the IDs made herein are used by addParents

Usage

```
addUIds(ped, format = getAutoIdFormat())
```

Arguments

ped	datatable that is the Pedigree. It contains pedigree information. The fields id, sire and dam are required.
format	sprintf template for the generated placeholder IDs; defaults to getAutoIdFormat() ("U%04d").

Details

The generated placeholder IDs default to the form Unnnn (a leading "U" plus a zero-padded integer), so they are alphanumeric and never contain a period ("."), honoring the ID rule enforced at data input by qcStudbook. The format is configurable via setAutoIdFormat (default "U%04d"). A candidate ID that is already in the pedigree is skipped, so a placeholder never duplicates the ID of a real animal.

Value

The updated pedigree in which each missing sire or dam of a single-parent record is replaced by a generated placeholder ID.

Examples

```
pedTwo <- data.frame(
  id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c("d0", "d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
  stringsAsFactors = FALSE
)
newPed <- addUIds(pedTwo)
newPed[newPed$id == "s1", ]
pedThree <-
  data.frame(
    id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
    sire = c("s0", "s0", "s4", NA, "s1", "s1", "s2", "s2"),
    dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
    sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
    stringsAsFactors = FALSE
  )
newPed <- addUIds(pedThree)
newPed[newPed$id == "s1", ]
```

---

alleleFreq *Count each allele in a vector*

Description

Part of Genetic Value Analysis

Usage

```
alleleFreq(alleles, ids = NULL)
```

Arguments

alleles	a numeric vector of alleles (integer codes) in the population
ids	character vector of IDs indicating to which animal each allele in alleles belongs. It must be the same length as alleles.

Details

If ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).

Value

A data.frame with columns allele (a factor) and freq. This is a table of allele counts within the population.

Examples

```
library(nprcgenekeepr)
data("ped1Alleles")
ids <- ped1Alleles$id
alleles <- ped1Alleles[, !(names(ped1Alleles) %in% c("id", "parent"))]
aF <- alleleFreq(alleles[[1]], ids = NULL)
aF[aF$freq >= 10, ]
```

---

appendColonySnapshot *Append a colony snapshot to a snapshot history*

Description

Pure merge for the longitudinal colony snapshot workflow (issue #167): takes a validated snapshot history (or NULL / a zero-row history for a first snapshot) and one new snapshot row, and returns the merged, date-ordered history. It writes nothing — script users persist the result themselves (e.g. write.csv(..., row.names = FALSE)), and the Shiny application writes only through a user-initiated download.

Usage

```
appendColonySnapshot(history, snapshot)
```

Arguments

history	validated snapshot history data.frame (see checkSnapshotHistory), or NULL / a zero-row history when recording the first snapshot.
snapshot	one-row data.frame holding the new snapshot, in the same schema as the history. The row count and columns are not checked; rows are bound to the history as given.

Details

The new snapshot must carry the same schemaVersion as the history, and its (snapshotDate, membershipRule) pair must not already be present — the pair identifies a snapshot uniquely. The same date under a different membership rule is legal.

Value

The merged history, ordered by snapshotDate.

Examples

```
history <- checkSnapshotHistory(readSnapshotHistory(system.file(
  "extdata", "examples", "example_snapshot_history.csv",
  package = "nprcgenekeepr"
)))
snapshot <- history[3L, ]
snapshot$snapshotDate <- as.Date("2026-07-15")
appendColonySnapshot(history, snapshot)
```

---

applyKinshipOverrides *Apply outside-information kinship overrides to a kinship matrix*

Description

Writes pairwise kinship coefficients from outside information into a computed kinship matrix, replacing the pedigree-derived value for the named pairs. Each (id1, id2, kinship) row sets both kmat[id1, id2] and its symmetric twin kmat[id2, id1]; all other cells are unchanged. This is a direct cell replacement – it does not propagate to descendant rows. A successful call reports the number of overrides applied with a message().

Usage

```
applyKinshipOverrides(kmat, overrides)
```

Arguments

kmat	a dense, symmetric, id-named numeric kinship matrix.
overrides	data.frame of overrides (id1, id2, kinship); NULL or a zero-row frame is a no-op returning kmat unchanged. Validated with checkKinshipOverrides.

Details

kmat must be a dense, symmetric, id-named base R matrix (the object kinship returns); a sparse Matrix object is out of contract. The function is strict: it stop()s on an id absent from the matrix and on a value above the exact positive-semi-definiteness bound sqrt(kmat[id1, id1] * kmat[id2, id2]). Soft, run-preserving handling of ids outside the analysis set is the caller's responsibility – reportGV warn-drops non-member ids before calling this. This function's own algorithm – writing outside-information overrides onto an already-computed matrix – is never applied inside kinship itself (it has several callers, including two simulations that must not take current-kinship overrides). This is a different kind of adjustment than kinship's own twinRelations parameter: a declared MZ-twin pair's genetic identity is a structural pedigree fact, like sire/dam/sex, not outside information overriding a pedigree-derived computation – both simulations pass a known twin pair through unchanged, so twin identity needs no shielding of the kind overrides require. See docs/planning/twin-relations-kinship-computation-plan.md §2.3.

Value

kmat with the override cells replaced (symmetric).

Examples

```
ped <- nprcgenekeepr::qcPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
overrides <- data.frame(
  id1 = ped$id[1], id2 = ped$id[2], kinship = 0.25,
  stringsAsFactors = FALSE
)
kmat <- applyKinshipOverrides(kmat, overrides)
```

---

appServer *Main Application Server for nprcgenekeepr*

Description

Server function for the main GeneKeepR Shiny application. Initializes all modules and manages shared reactive state between them.

Usage

```
appServer(input, output, session)
```

Arguments

input	Shiny input object
output	Shiny output object
session	Shiny session object

Details

The server handles:

- Configuration loading from site-specific config files
- Navigation button handlers for the home page
- Dynamic tab management for QC errors and changed columns
- Module initialization and data flow between modules

Value

No return value, called for side effects. As a 'Shiny' server function, `appServer()` is invoked by the 'Shiny' runtime to wire up the application's reactive outputs, observers, and module servers for a running GeneKeepR session.

See Also

`appUI` for the corresponding UI function

`modInputServer` for data input module

`modPedigreeServer` for pedigree browser module

`modGeneticValueServer` for genetic value analysis

`shouldShowChangedColsTab` for changed columns tab logic

`loadSiteConfig` for site configuration loading

---

appUI *Main Application UI for nprcgenekeepr*

Description

Main Application UI for nprcgenekeepr

Usage

```
appUI(siteInfo = NULL)
```

Arguments

`siteInfo`	Named list of site configuration as returned by `getSiteInfo`; defaults to `NULL`, in which case it is resolved internally via `getSiteInfo(expectConfigFile = FALSE)`. A present-but-malformed site-config file makes that call fail; the failure is caught and logged (`futile.logger::flog.warn`) rather than propagating, and the UI falls back to hiding the ORIP Reporting tab. Its `center` and `configFile` elements gate the Oregon (ONPRC)-specific ORIP Reporting tab, which is shown only for an actual ONPRC configuration (see `shouldShowOripTab`).

Value

A `shiny.tag.list` object (as produced by `shiny::tagList()`) describing the complete GeneKeepR user interface; pass it as the `ui` argument to `shiny::shinyApp()` or `shiny::runApp()`.

---

assignAlleles *Assign parent alleles randomly*

Description

Assign parent alleles randomly

Usage

```
assignAlleles(alleles, parentType, parent, id, n)
```

Arguments

`alleles`	a list with a list `alleles$alleles`, which is a list of list containing the alleles for each individual's sire and dam that have been assigned thus far and `alleles$counter` that is the counter used to track the lists of`alleles$alleles`.
`parentType`	character vector of length one with value of `"sire"` or `"dam"`.
`parent`	either `ped[id, "sire"]` or `ped[id, "dam"]`.
`id`	character vector of length one containing the animal ID
`n`	integer indicating the number of iterations to simulate. It is used only when `parent` is `NA`; for a known parent the number of iterations is taken from that parent's already assigned alleles. If a known parent has no alleles assigned yet the function stops with the message "sire and dam must have had alleles assigned: logic error".

Value

The original list `alleles` passed into the function with newly randomly assigned alleles to each `id` based on dam and sire genotypes.

Examples

```
alleles <- list(alleles = list(), counter = 1)
alleles <- assignAlleles(alleles,
  parentType = "sire", parent = NA,
  id = "o1", n = 4
)
alleles
alleles <- assignAlleles(alleles,
  parentType = "dam", parent = NA,
  id = "o1", n = 4
)
alleles
```

`backfillColonySnapshots` *Rebuild earlier colony snapshots from a pedigree's birth and exit dates*

Description

Builds one Genetic-Health Trends snapshot for each past date from the birth and exit dates already in a pedigree, so a trend can be drawn without waiting seasons for recorded snapshots (issue #170). Each row is the same 27-column row `createColonySnapshot` makes and passes `checkSnapshotHistory`, filed under its own membership rule, `"reconstructedLiving"`, so rebuilt rows form a line of their own and never mix with recorded ones.

Usage

```
backfillColonySnapshots(
  ped,
  dates = NULL,
  guIter,
  guThresh,
  from = NULL,
  to = NULL,
  by = "year",
  existingHistory = NULL,
  minAnimals = 2L,
  updateProgress = NULL,
  ...
)
```

Arguments

`ped`	A quality-controlled pedigree data.frame (see `qcStudbook`) holding at least `id`, `sire`, `dam`, `sex`, `gen`, `birth` and `exit` (the last two as `Date` columns), and a `status` column when the studbook has one. Pass the full pedigree, not one already trimmed to a focal list of animals.
`dates`	The dates to rebuild: `Date` values or ISO-8601 (YYYY-MM-DD) strings, at least one, with no missing or duplicated value. Leave it out (`NULL`) to build the series described by `from`, `to` and `by`; it cannot be combined with them.
`guIter`	Single positive whole number: the number of gene-drop iterations, recorded in each row. Required — there is no default, so the recorded settings cannot silently differ from the ones used.
`guThresh`	Single positive whole number: the genetic uniqueness threshold, recorded in each row. Required.
`from`	The first date of the series: a `Date` or an ISO-8601 string. The default is the first 1 January with at least `minAnimals` living animals.
`to`	The last date of the series, as `from`. The default is the last 1 January on or before the latest birth or exit date in `ped`.

by	The step of the series: a string for seq.Date, such as "year" (the default), "6 months" or "quarter".
existingHistory	A snapshot history to add to (see checkSnapshotHistory), or NULL for none. Its rows are kept as they are, with any extra columns, and a date whose "reconstructedLiving" row is already there is not rebuilt. A row of another membership rule on the same date does not count as already there.
minAnimals	Single whole number, at least 2 (the least reportGV can analyze): the fewest living animals a 1 January needs to start the default series. Default 2.
updateProgress	Function or NULL. If it is a function, it is called once for each date, after the date has been rebuilt, skipped or has failed, with the arguments n (the number of dates in the run), value (how many are done) and detail (the date as text), to update a shiny::Progress object. It is not passed on to reportGV.
...	Further arguments passed to reportGV (for example kinshipOverrides or twinRelations, the latter cut to each date as described above).

Details

The dates. Give dates, or leave it out for a series of by steps (one year by default) from from to to. Unless you give them, the series starts at the first 1 January with at least minAnimals living animals and ends at the last 1 January on or before the latest birth or exit date in the pedigree. minAnimals only moves that default start: a date you name, in dates or as from, is never refused for having few animals.

Who was alive. For each date the pedigree is copied and every exit date after that date is blanked, because the demographic effective-size estimators read the exit column as it stands (see calcNeSexRatio): without the blanking an animal that was alive on the date but has left since would not count as a living breeder. The population is the real animals alive on the date. An animal is alive from its birth date; one with no birth date is alive from the birth of its first offspring that has a date (it was born before that offspring). It is alive until its exit date. An animal with no exit date counts only when the studbook lists it as living, the test the application applies today (status ALIVE, or no exit date when the pedigree has no status column), so an animal whose status is UNKNOWN never counts. Generated stand-in parents never enter the population. The pedigree is trimmed to that population and its ancestors (trimPedigree), analyzed with reportGV, and summarized with createColonySnapshot. For the date a pedigree was exported, the row equals the snapshot the application records for that pedigree.

What each date rests on. The support table says how many animals were counted, how many of them were placed through an offspring, how many more the studbook does not rule out (animals with no birth date that are not counted and had not left), and how much of the parentage is recorded. No threshold refuses a date; read these figures beside the trend. Where few sires are recorded the sex-ratio effective size reads low because of the missing records, not because of the colony.

Dates that fail. A date that cannot be analyzed (for example one on which fewer than two animals were alive) does not stop the others. It is listed in skipped with the message of the error, and one warning says how many dates failed. When no date can be built, the function stops and names the dates. A date whose "reconstructedLiving" row is already in existingHistory is not rebuilt; it is listed in skipped without a warning.

Twin pairs. A twinRelations table (see checkTwinRelations) is used on each date as far as that date's animals allow. Declared identical ("MZ twin") animals form groups, as in kinship, so

the pairs A-B and B-C make the group A, B and C. On a date the groups are cut to the animals in that date's pedigree, which holds the animals alive then and their ancestors; a group left with two or more animals is applied and the others are left out. An animal missing from the pedigree on a date cannot change anyone's kinship that day, so a twin born later, or one who left and has no descendants, no longer makes the date fail. An identifier that is in no pedigree at all is a mistake and stops the function before any date is built.

Limits: today's parentage and today's genotypes are used for every date, so a parent identified later is treated as known earlier; the effective sizes follow the definition of a living breeder in `calcNeSexRatio`; and two close dates can differ only by gene-drop sampling.

Value

A list of three data.frames. `history`: the rebuilt snapshots (and the rows of `existingHistory`, if given), in date order, in the 27-column snapshot-history schema. `support`: one row for each date in the run, with `snapshotDate`, `nLiving` (the animals counted), `nPlacedByOffspring` (of those, the ones with no birth date), `nPossiblyAlive`, `pctSireKnown` and `pctDamKnown` (the percentage of the living animals whose sire or dam is recorded, that is neither blank nor a generated stand-in; `NA` when nobody was alive). `skipped`: the dates not rebuilt, with `snapshotDate` and `reason`.

See Also

`createColonySnapshot`, `appendColonySnapshot`, `checkSnapshotHistory`

Examples

```
ped <- nprcgenekeepr::qcPed
result <- backfillColonySnapshots(ped,
  dates = as.Date(c("2005-01-01", "2010-01-01")),
  guIter = 10L, guThresh = 1L
)
result$history[, c("snapshotDate", "membershipRule", "nAnimals")]
result$support

## a yearly series, added to the history above: the dates already there
## are not rebuilt
series <- backfillColonySnapshots(ped,
  from = "2005-01-01", to = "2008-01-01",
  guIter = 10L, guThresh = 1L, existingHistory = result$history
)
series$history[, c("snapshotDate", "nAnimals")]
series$skipped
```

`buildMarkerGenotypeMatrix` *Pivot a long-format marker genotype table into a wide genotype matrix*

Description

Converts a validated long-format marker genotype table (see checkMarkerGenotypeFile) into a wide id x locus character matrix, the input shape markerKinship consumes.

Usage

```
buildMarkerGenotypeMatrix(genotype)
```

Arguments

genotype	dataframe with long-format marker genotype data, as returned by checkMarkerGenotypeFile: columns id, locus, allele1, allele2.

Details

Row and column order follow first appearance in genotype, not a string sort – a string sort would place "L10" before "L2" for any panel with more than nine loci, silently scrambling locus order. If genotype has more than one row for the same id x locus, the last row silently overwrites the others, so run checkMarkerGenotypeFile first.

Value

A character matrix with one row per unique id and one column per unique locus. Each cell holds that individual's two alleles at that locus, sorted (by string collation, so locale-dependent for mixed-case alleles) and joined by "/" (e.g. "A/B"), or NA when that individual has no genotype record at that locus. A record with a missing allele gives NA, the same as a record that is not in the file; checkMarkerGenotypeFile does not reject such rows.

See Also

checkMarkerGenotypeFile, markerKinship

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("A", "A", "B", "B"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("A", "B", "B", "B"),
  stringsAsFactors = FALSE
)
buildMarkerGenotypeMatrix(markerGenotype)
```

`calcA` *Count each individual's rare alleles per simulation*

Description

Part of Genetic Value Analysis

Usage

```
calcA(alleles, threshold = 1L, byID = FALSE)
```

Arguments

`alleles`	a matrix with {V1 ... Vn, id, parent} providing the alleles an animal received during each simulation. The first n columns provide the alleles; the final two columns provide the animal ID and the parent the allele came from.
`threshold`	an integer indicating the maximum number of copies of an allele that can be present in the population for it to be considered rare. Default is 1.
`byID`	logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).

Value

A matrix with named rows indicating the number of rare alleles (0, 1 or 2) an animal had during each round of simulation (indicated in columns).

References

Ballou JD, Lacy RC. 1995. Identifying genetically important individuals for management of genetic variation in pedigreed populations, p 77-111. In: Ballou JD, Gilpin M, Foose TJ, editors. Population management for survival and recovery. New York (NY): Columbia University Press.

MacCluer JW, et al. 1986. Pedigree analysis by computer simulation. Zoo Biology 5:147-160.

See Also

Other genetic value analysis: `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcKurtosis()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`, `calcSkewness()`

Examples

```
library(nprcgenekeepr)
rare <- calcA(nprcgenekeepr::ped1Alleles, threshold = 3, byID = FALSE)
```

calcAge *Calculate animal ages*

Description

Part of Pedigree Curation

Usage

```
calcAge(birth, exit)
```

Arguments

birth	Date vector of birth dates
exit	Date vector of exit dates.

Details

Given vectors of birth and exit dates, calculate an individuals age. If no exit date is provided, the calculation is based on the current date.

Value

A numeric vector (NA allowed) indicating age in years, rounded to one decimal place, from "birth" to "exit" or the current date if "exit" is NA. If birth has length zero, it is returned unchanged (a Date, not numeric, vector).

Examples

```
library(nprcgenekeepr)
qcPed <- nprcgenekeepr::qcPed
originalAge <- qcPed$age ## ages calculated at time of data collection
currentAge <- calcAge(qcPed$birth, qcPed$exit) ## assumes no changes in
## colony
```

calcFE *Calculate founder equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFE(ped)
```

Arguments

ped	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population).

Details

The pedigree must have no partial parentage (every animal has both parents known or both unknown); calcFE stops with an error otherwise.

Value

The founder equivalents FE = 1 / sum(p ^ 2), where p is the vector of founder mean contributions to the current descendants.

References

Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: calcA(), calcFEFG(), calcFG(), calcFGSE(), calcGU(), calcGUSE(), calcGeneDiversity(), calcKurtosis(), calcNeSexRatio(), calcNeVariance(), calcRetention(), calcSkewness()

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
ped$population <- getGVPopulation(ped, NULL)
pedFactors <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = TRUE
)
pedFactors["gen"] <- findGeneration(
  pedFactors$id, pedFactors$sire,
  pedFactors$dam
)
pedFactors$population <- getGVPopulation(pedFactors, NULL)
fe <- calcFE(ped)
```

```
feFactors <- calcFE(pedFactors)
```

`calcFEFG` *Calculate founder equivalents and founder genome equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFEFG(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). The pedigree must have no partial parentage (every animal has both parents known or both unknown); `calcFEFG` stops with an error otherwise. Every animal with both parents unknown is treated as a separate, unrelated founder, including an animal whose id was generated to stand in for an unknown parent. Such a placeholder may in fact be a known founder, another animal in the population, or an animal outside it, so the unrelated-founder assumption can overstate `FE` and `FG` when placeholders are common. `reportGV` lists only the *known* founders, so its founder counts can be smaller than the founder set used here.
`alleles`	dataframe contains an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

The list containing the founder equivalents, `FE = 1 / sum(p ^ 2)`, and the founder genome equivalents, `FG = 1 / sum( (p ^ 2) / r)` where `p` is the vector of founder mean contributions to the current descendants and `r` is the mean number of founder alleles retained in the gene dropping experiment.

`FE` is deterministic and always returned. `FG` is `NA` (with a warning) when a contributing founder (`p > 0`) is retained in zero of the gene-drop iterations (`r == 0`), which would otherwise collapse `FG` silently to 0; raise the number of iterations. See `calcFGSE` for the sampling standard error of `FG`.

References

Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcKurtosis()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`, `calcSkewness()`

Examples

```
data(lacy1989Ped)
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy

library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
alleles <- lacy1989PedAlleles
pedFactors <- data.frame(
  id = as.factor(ped$id),
  sire = as.factor(ped$sire),
  dam = as.factor(ped$dam),
  gen = ped$gen,
  population = ped$population,
  stringsAsFactors = TRUE
)
allelesFactors <- geneDrop(pedFactors$id, pedFactors$sire, pedFactors$dam,
  pedFactors$gen,
  genotype = NULL, n = 1000,
  updateProgress = NULL
)
feFg <- calcFEFG(ped, alleles)
feFgFactors <- calcFEFG(pedFactors, allelesFactors)
```

`calcFG` *Calculate founder genome equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFG(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). The pedigree must have no partial parentage (every animal has both parents known or both unknown); `calcFG` stops with an error otherwise.
`alleles`	dataframe contains an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

The founder genome equivalents, `FG = 1 / sum( (p ^ 2) / r)` where `p` is the vector of founder mean contributions to the current descendants and `r` is the mean number of founder alleles retained in the gene dropping experiment.

Returns `NA` with a warning when a contributing founder (`p > 0`) is retained in zero of the gene-drop iterations (`r == 0`): that term is `p^2 / 0 = Inf`, which would otherwise collapse `FG` silently to 0. Raise the number of iterations. See `calcFGSE` for the sampling standard error of `FG`.

References

Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcKurtosis()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`, `calcSkewness()`

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy

library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
ped$population <- getGVPopulation(ped, NULL)
pedFactors <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = TRUE
)
pedFactors["gen"] <- findGeneration(
  pedFactors$id, pedFactors$sire,
  pedFactors$dam
)
pedFactors$population <- getGVPopulation(pedFactors, NULL)
alleles <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = NULL,
  n = 1000, updateProgress = NULL
)
allelesFactors <- geneDrop(pedFactors$id, pedFactors$sire, pedFactors$dam,
  pedFactors$gen,
```

```
  genotype = NULL, n = 1000,
  updateProgress = NULL
)
fg <- calcFG(ped, alleles)
fgFactors <- calcFG(pedFactors, allelesFactors)
```

`calcFGSE` *Calculate the standard error of founder genome equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFGSE(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). The pedigree must have no partial parentage (every animal has both parents known or both unknown); `calcFGSE` stops with an error otherwise.
`alleles`	dataframe containing an `AlleleTable`: one column per gene-drop iteration, followed by an `id` column and a `parent` column. Produced by `geneDrop()`; the same input `calcFG` takes.

Details

Founder genome equivalents (`calcFG`) is a Monte Carlo estimate: `FG = 1 / sum(p^2 / r)`, where the founder contributions `p` are deterministic but the mean allelic retention values `r` (`calcRetention`) are averages over the gene-drop iterations, so `FG` carries sampling error that shrinks as the number of iterations grows. Unlike genome uniqueness (a mean, whose standard error is a column variance), `FG` is a nonlinear function of `r`, so its standard error is obtained by the delta method (first-order linearization).

With `S = sum(p_f^2 / r_f)` and `FG = 1 / S`, the gradient is `dFG/dr_f = FG^2 * p_f^2 / r_f^2`. Writing `R` for the founder-by- iteration retention matrix (each column an independent gene drop), the influence series `y_k = sum_f (dFG/dr_f) * R[f, k]` has `sd(y) / sqrt(K)` equal to the full delta-method standard error, including the within-iteration covariance among founders. This influence form is used because it folds in that covariance automatically and never forms the founder-by-founder covariance matrix.

Founders are matched between `p` and `r` by name (not position), so the result is correct even when the founders are not in sorted pedigree order.

A contributing founder (`p > 0`) that is retained in zero of the iterations (`r == 0`) makes `FG` undefined (the same degeneracy that `calcFG` reports as `NA`); in that case this function returns `NA` with a warning advising more iterations. Founders that do not contribute to the current population (`p == 0`) are dropped, so the standard error refers to exactly the founder set `FG` is computed from.

Value

A single numeric value: the Monte Carlo sampling standard error of the colony founder-genome-equivalent estimate, on the same scale as `calcFG`. `NA` (with a warning) when a contributing founder has zero retention.

See Also

`calcFG`, `calcFEFG`, `calcRetention`, `calcGUSE`, `reportGV`

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcKurtosis()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`, `calcSkewness()`

Examples

```
library(nprcgenekeepr)
data("lacy1989Ped")
data("lacy1989PedAlleles")
calcFGSE(lacy1989Ped, lacy1989PedAlleles)
```

---

calcGeneDiversity *Calculate gene diversity from founder genome equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcGeneDiversity(fg)
```

Arguments

`fg`	Founder genome equivalents scalar, as returned by `calcFG()` or the `$FG` element of `calcFEFG()`. `NA` yields `NA`.

Details

Gene diversity is the expected heterozygosity retained relative to the founding gene pool, `GD = 1 - 1 / (2 * FG)`, where `FG` is the founder genome equivalents (see `calcFG`). It summarizes how much of the founders' allelic diversity still survives: it is 0 when `FG = 0.5` (the smallest value `FG` takes in practice) and approaches (never reaches) 1 as `FG` grows.

`GD` is a diversity proportion, not a count of effective individuals, and it is computed over the same analysis set as `FG`. `NA` propagates: when `FG` is `NA` (the zero-retention degeneracy `calcFG()` reports), `GD` is `NA`.

Value

The gene diversity `GD = 1 - 1 / (2 * fg)`: a single number below 1 (realized values lie in `[0, 1)`), or `NA` when `fg` is `NA`.

References

Gene diversity is derived here from the founder genome equivalents (`calcFG`) of Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcKurtosis()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`, `calcSkewness()`

Examples

```
calcGeneDiversity(20) # 0.975
calcGeneDiversity(52.75) # gene diversity at the qcPed FG
```

---

calcGU *Calculate genome uniqueness for each population ID*

Description

Part of Genetic Value Analysis

Usage

```
calcGU(alleles, threshold = 1L, byID = FALSE, pop = NULL)
```

Arguments

`alleles`	dataframe of containing an `AlleleTable`. This is a table of allele information produced by `geneDrop()`. An AlleleTable contains information about alleles an ego has inherited. It contains the following columns: • `V1` — Unnamed integer column representing allele 1. • `V2` — Unnamed integer column representing allele 2. • `...` — Unnamed integer columns representing alleles. • `Vn` — Unnamed integer column representing the nth column. • `id` — A character vector of IDs for a set of animals. • `parent` — A character vector with values of sire and dam.
`threshold`	an integer indicating the maximum number of copies of an allele that can be present in the population for it to be considered rare. Default is 1.

byID	logical variable of length 1 that is passed through to eventually be used by alleleFreq(), which calculates the count of each allele in the provided vector. If byID is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).
pop	character vector with animal IDs to consider as the population of interest, otherwise all animals will be considered. The default is NULL.

Details

The following functions calculate genome uniqueness according to the equation described in Ballou & Lacy.

It should be noted, however that this function differs slightly in that it does not distinguish between founders and non-founders in calculating the statistic.

Ballou & Lacy describe genome uniqueness as "the proportion of simulations in which an individual receives the only copy of a founder allele." We have interpreted this as meaning that genome uniqueness should only be calculated for living, non-founder animals. Alleles possessed by living founders are not considered when calculating genome uniqueness.

We have a differing view on this, since a living founder can still contribute to the population. The function below calculates genome uniqueness for all living animals and considers all alleles. It does not ignore living founders and their alleles.

Our results for genome uniqueness will, therefore differ slightly from those returned by Pedscope. Pedscope calculates genome uniqueness only for non-founders and ignores the contribution of any founders in the population. This will cause Pedscope's genome uniqueness estimates to possibly be slightly higher for non-founders than what this function will calculate.

The estimates of genome uniqueness for founders within the population calculated by this function should match the "founder genome uniqueness" measure calculated by Pedscope.

Note that calcGU computes this statistic for every animal and still includes living founders' alleles; it is unchanged. The genetic value report (reportGV) separately applies a colony decline-to-credit policy that reports genome uniqueness as 0 for unknown-origin both-unknown ("Undetermined") animals.

Value

Dataframe rows: id, col: gu A single-column table of genome uniqueness values as percentages. Rownames are set to 'id' values that are part of the population.

References

Ballou JD, Lacy RC. 1995. Identifying genetically important individuals for management of genetic variation in pedigreed populations, p 77-111. In: Ballou JD, Gilpin M, Foose TJ, editors. Population management for survival and recovery. New York (NY): Columbia University Press.

MacCluer JW, et al. 1986. Pedigree analysis by computer simulation. Zoo Biology 5:147-160.

See Also

Other genetic value analysis: calcA(), calcFE(), calcFEFG(), calcFG(), calcFGSE(), calcGUSE(), calcGeneDiversity(), calcKurtosis(), calcNeSexRatio(), calcNeVariance(), calcRetention(), calcSkewness()

Examples

```
library(nprcgenekeepr)
ped1Alleles <- nprcgenekeepr::ped1Alleles
gu_1 <- calcGU(ped1Alleles, threshold = 1, byID = FALSE, pop = NULL)
gu_2 <- calcGU(ped1Alleles, threshold = 3, byID = FALSE, pop = NULL)
gu_3 <- calcGU(ped1Alleles,
  threshold = 3, byID = FALSE,
  pop = ped1Alleles$id[20:60]
)
```

`calcGUSE` *Calculate the standard error of genome uniqueness*

Description

Part of Genetic Value Analysis

Usage

```
calcGUSE(alleles, threshold = 1L, byID = FALSE, pop = NULL)
```

Arguments

`alleles`	dataframe containing an `AlleleTable` (the same input `calcGU` takes): one integer column per gene-drop iteration, followed by an `id` column and a `parent` column. Produced by `geneDrop()`.
`threshold`	an integer indicating the maximum number of copies of an allele that can be present in the population for it to be considered rare. Default is 1.
`byID`	logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).
`pop`	character vector with animal IDs to consider as the population of interest, otherwise all animals will be considered. The default is NULL.

Details

Genome uniqueness (`calcGU`) is a Monte Carlo estimate: it is the average, over the gene-drop iterations, of the proportion of an animal's two allele copies that are population-rare. Because it is an average over independent simulated iterations, it carries sampling error that shrinks as the number of iterations grows.

For animal `i` let `m_ik = rare[i, k] / 2` be the per-iteration value in iteration `k` (so the mean of `m_ik` over the `K` iterations equals `gu_i / 100`). This function returns the exact Monte Carlo standard error of that mean, on the same percentage scale as `calcGU`:

$$guSE_i = 100 \times \sqrt{\frac{var(m_{i\cdot})}{K}}$$

The standard error is computed from the same per-iteration rare-allele matrix (calcA) that calcGU averages, so it is correct for any threshold / byID without a closed-form approximation. An animal whose rare-allele count does not vary across iterations has a standard error of 0.

Value

Dataframe rows: id, col: guSE A single-column table of genome-uniqueness standard errors as percentages. Rownames are set to 'id' values that are part of the population.

See Also

calcGU, calcA, reportGV

Other genetic value analysis: calcA(), calcFE(), calcFEFG(), calcFG(), calcFGSE(), calcGU(), calcGeneDiversity(), calcKurtosis(), calcNeSexRatio(), calcNeVariance(), calcRetention(), calcSkewness()

Examples

```
library(nprcgenekeepr)
ped1Alleles <- nprcgenekeepr::ped1Alleles
guSE <- calcGUSE(ped1Alleles, threshold = 3, byID = FALSE, pop = NULL)
```

---

calcKurtosis *Calculate bias-adjusted sample excess kurtosis*

Description

Part of the Genetic Value Analysis

Usage

```
calcKurtosis(x, na.rm = TRUE)
```

Arguments

x	A numeric vector.
na.rm	Logical; remove NA values before computing. Default TRUE.

Details

The bias-adjusted Fisher-Pearson standardized EXCESS kurtosis coefficient, G2 = ((n + 1) * g2 + 6) * (n - 1) / ((n - 2) * (n -3)), where g2 = m4 / m2^2 - 3 and m2/m4 are the second/fourth central sample moments of x – the "Method 2" adjustment of Joanes and Gill (1998), the same convention SPSS, SAS, and Excel report by default and the type = 2 option in the e1071 CRAN package. Excess (not raw) kurtosis: a normal distribution reads 0. Positive values indicate heavier tails / a sharper peak than normal; negative, lighter tails / a flatter peak.

Returns NA when na.rm = FALSE and x contains an NA, when x has fewer than 4 non-NA values (n <= 3, the adjustment term divides by (n - 2) * (n - 3)) or has zero variance (all remaining values identical) – both would otherwise divide by zero.

Value

The bias-adjusted sample excess kurtosis, a single number; NA when degenerate (see Details).

References

Joanes, D.N. and Gill, C.A. (1998) "Comparing measures of sample skewness and kurtosis." *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(1), 183-189.

See Also

calcSkewness

Other genetic value analysis: calcA(), calcFE(), calcFEFG(), calcFG(), calcFGSE(), calcGU(), calcGUSE(), calcGeneDiversity(), calcNeSexRatio(), calcNeVariance(), calcRetention(), calcSkewness()

Examples

```
calcKurtosis(c(1, 2, 3, 4, 100)) # 4.986866, heavy-tailed
calcKurtosis(c(1, 2, 3, 4, 5)) # -1.2, flatter than normal
```

---

calcNeSexRatio *Calculate the demographic sex-ratio effective population size*

Description

Part of the Genetic Value Analysis

Usage

```
calcNeSexRatio(ped)
```

Arguments

ped	Pedigree data.frame with id, sire, dam, and sex; exit is used to identify living animals when present.

Details

The sex-ratio effective size, Ne = 4 * nMale * nFemale / (nMale + nFemale), is the effective population size implied by an unequal breeding sex ratio, where nMale and nFemale are the numbers of current living breeders that are known male and known female. It quantifies the diversity lost when many of one sex are bred to few of the other (as in a harem colony): it equals the census count when the sexes are balanced and falls toward four times the rarer sex as the ratio skews.

The breeders are the current living breeders of ped (living animals that appear as a sire or dam, excluding auto-generated unknown parents), independent of which animals are selected as probands – a different population than the analysis-set founder statistics (calcFE, calcFG, calcGeneDiversity). Only animals with a known sex ("M" or "F") are counted; unknown and hermaphrodite breeders are

excluded. When either sex is absent among the living breeders (`nMale == 0` or `nFemale == 0`, including no living breeders at all), the result is `0`: a single breeding sex contributes no diversity from sex balance.

Like all effective-size estimators this idealizes a Wright-Fisher population (constant size, discrete generations, random union of gametes within each sex); a managed colony departs from those assumptions, so read the result as a sex-ratio index rather than a literal head count.

Value

The sex-ratio effective size, a single non-negative number; `0` when either breeding sex is absent among the living breeders.

References

Crow, J. F. and Kimura, M. (1970) *An Introduction to Population Genetics Theory*. Harper and Row, New York.

See Also

`calcGeneDiversity`

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcKurtosis()`, `calcNeVariance()`, `calcRetention()`, `calcSkewness()`

Examples

```
ped <- data.frame(
  id = c("s1", "d1", "d2", "k1", "k2"),
  sire = c(NA, NA, NA, "s1", "s1"),
  dam = c(NA, NA, NA, "d1", "d2"),
  sex = c("M", "F", "F", "M", "F"),
  exit = c(NA, NA, NA, NA, NA),
  stringsAsFactors = FALSE
)
calcNeSexRatio(ped) # 4 * 1 * 2 / (1 + 2) = 2.666667
```

---

calcNeVariance *Calculate the variance effective population size*

Description

Part of the Genetic Value Analysis

Usage

```
calcNeVariance(ped)
```

Arguments

ped	Pedigree data.frame with id, sire, and dam; exit is used to identify living animals when present.

Details

The variance effective size measures the diversity lost to unequal family sizes – typically the dominant reducer of effective size in a harem colony, where a few breeders produce most of the offspring. It is the mean-adjusted Crow & Kimura (1970) form

$$N_e = \frac{N\bar{k} - 1}{\bar{k} - 1 + V_k/\bar{k}}$$

where N is the number of current living breeders, $\bar{k}$ the mean number of lifetime offspring among them, and V_k the variance of those offspring counts. This general form makes no constant-size assumption and reduces to the classic `(4N - 2) / (Vk + 2)` at exact replacement ($\bar{k} = 2$); it is preferred over that bare form, which assumes $\bar{k} \approx 2$ and misstates the effective size when the mean family size departs from replacement.

The breeders are the current living breeders of ped (living animals that appear as a sire or dam, excluding auto-generated unknown parents), independent of which animals are selected as probands – a different population than the analysis-set founder statistics (calcFE, calcFG, calcGeneDiversity). Unlike the sex-ratio effective size (calcNeSexRatio), breeders of every sex are counted. When fewer than two living breeders are present the variance is undefined and the result is NA.

Like all effective-size estimators this idealizes a Wright-Fisher population (constant size, discrete generations, random union of gametes); a managed colony departs from those assumptions, so read the result as a family-size-variance index rather than a literal head count.

Value

The variance effective size, a single number; NA when there are fewer than two living breeders.

References

Crow, J. F. and Kimura, M. (1970) *An Introduction to Population Genetics Theory*. Harper and Row, New York.

See Also

calcNeSexRatio, calcGeneDiversity

Other genetic value analysis: calcA(), calcFE(), calcFEFG(), calcFG(), calcFGSE(), calcGU(), calcGUSE(), calcGeneDiversity(), calcKurtosis(), calcNeSexRatio(), calcRetention(), calcSkewness()

Examples

```
ped <- data.frame(
  id = c("s1", "d1", "k1", "k2", "k3"),
  sire = c(NA, NA, "s1", "s1", "s1"),
```

```
  dam = c(NA, NA, "d1", "d1", "d1"),
  sex = c("M", "F", "M", "F", "F"),
  exit = c(NA, NA, NA, NA, NA),
  stringsAsFactors = FALSE
)
calcNeVariance(ped) # 2 breeders, equal families: (2*3-1)/(3-1) = 2.5
```

`calcRetention` *Calculate allelic retention*

Description

Part of Genetic Value Analysis

Usage

```
calcRetention(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, population). It is assumed that the pedigree has no partial parentage. When the `population` column is missing or has no `TRUE` values, there are no descendants and every retention value is 0.
`alleles`	dataframe of containing an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

A named one-dimensional array with one value per founder: the proportion of the gene-drop simulations (a number from 0 to 1, not a count) in which the founder's allele is retained in the population's descendants.

References

Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcKurtosis()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcSkewness()`

Examples

```
library(nprcgenekeepr)
data("lacy1989Ped")
data("lacy1989PedAlleles")
ped <- lacy1989Ped
alleles <- lacy1989PedAlleles
retention <- calcRetention(ped, alleles)
```

`calcSkewness` *Calculate bias-adjusted sample skewness*

Description

Part of the Genetic Value Analysis

Usage

```
calcSkewness(x, na.rm = TRUE)
```

Arguments

`x`	A numeric vector.
`na.rm`	Logical; remove `NA` values before computing. Default `TRUE`.

Details

The bias-adjusted Fisher-Pearson standardized skewness coefficient, `G1 = g1 * sqrt(n * (n - 1)) / (n - 2)`, where `g1 = m3 / m2^1.5` and `m2`/`m3` are the second/third central sample moments of `x` – the "Method 2" adjustment of Joanes and Gill (1998), the same convention SPSS, SAS, and Excel report by default and the `type = 2` option in the `e1071` CRAN package. A positive value indicates a longer right tail; negative, a longer left tail; `0`, a symmetric distribution.

Returns `NA` when `na.rm = FALSE` and `x` contains an `NA`, when `x` has fewer than 3 non-`NA` values (`n <= 2`, the adjustment term divides by `n - 2`) or has zero variance (all remaining values identical) – both would otherwise divide by zero.

Value

The bias-adjusted sample skewness, a single number; `NA` when degenerate (see Details).

References

Joanes, D.N. and Gill, C.A. (1998) "Comparing measures of sample skewness and kurtosis." *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(1), 183-189.

See Also

calcKurtosis

Other genetic value analysis: calcA(), calcFE(), calcFEFG(), calcFG(), calcFGSE(), calcGU(), calcGUSE(), calcGeneDiversity(), calcKurtosis(), calcNeSexRatio(), calcNeVariance(), calcRetention()

Examples

```
calcSkewness(c(1, 2, 3, 4, 100)) # 2.232396, a long right tail
calcSkewness(c(1, 2, 3, 4, 5)) # 0, symmetric
```

---

calcSnapshotDeltas *Compare two colony snapshots metric by metric*

Description

Computes per-metric deltas between two snapshots of a longitudinal colony snapshot history (issue #167). The two snapshots must belong to the same membership-rule series: a snapshot is identified by its (snapshotDate, membershipRule) pair, and trend comparisons are only meaningful like with like. The result carries one row per numeric schema column — the 18 metric columns plus the 3 composition counts (nAnimals, nMales, nFemales), whose deltas make membership churn between the two snapshots visible.

Usage

```
calcSnapshotDeltas(history, from, to, membershipRule = NULL)
```

Arguments

history	data.frame holding the snapshot history in the 27-column version-1 schema; validated internally with checkSnapshotHistory.
from	character or Date; the snapshotDate of the baseline snapshot.
to	character or Date; the snapshotDate of the comparison snapshot.
membershipRule	character; the membership-rule series the two dates belong to. The default NULL resolves automatically when the history holds a single rule and stop()s, naming the rules present, when it holds several.

Details

The comparabilityFlag column surfaces provenance differences between the two snapshots rather than refusing them: it is NA_character_ when the snapshots are comparable and otherwise names each differing provenance field with both values. A guIter difference flags the gene-drop-derived metrics (fg, fgSE, neGD, and the gu aggregates); a guThresh difference flags only the gu aggregates (the threshold reaches only the genome-uniqueness computation); a packageVersion difference flags every row.

Value

A data.frame with columns `metric`, `from`, `to`, `delta` (`to` - `from`), and `comparabilityFlag`, one row per numeric schema column in schema order.

Examples

```
history <- checkSnapshotHistory(readSnapshotHistory(system.file("extdata",
  "examples", "example_snapshot_history.csv",
  package = "nprcgenekeepr"
)))
deltas <- calcSnapshotDeltas(history,
  from = "2025-01-15", to = "2025-07-15",
  membershipRule = "wholePedigree"
)
```

---

calculateSexRatio *Calculate the sex ratio of a set of animals*

Description

The Males are counted when the `ped$sex` value is `"M"`. Females are counted when the `ped$sex` value is not `"M"`. This means animals with ambiguous sex are counted with the females.

Usage

```
calculateSexRatio(ids, ped, additionalMales = 0L, additionalFemales = 0L)
```

Arguments

`ids`	character vector of animal IDs
`ped`	dataframe that is the `Pedigree`. It contains pedigree information including the IDs listed in `ids`.
`additionalMales`	Integer value of males to add to those within the group when calculating the ratio. The additions always count, except when `ids` has no animals or no males: with no males, the ratio is `Inf` unless `additionalMales` is greater than 0 (then `additionalFemales` still counts); with no animals, see *Value*. Default is 0.
`additionalFemales`	Integer value of females to add to those within the group when calculating the ratio. The additions always count, except as described for `additionalMales`. Default is 0.

Value

Numeric value of the sex ratio of the animals provided, expressed as the number of non-males per male. It is `Inf` when there are no males (and no males are added) but at least one non-male; `0` when `ids` is empty, no females are added and males are added; and `NA` when `ids` is empty and nothing is added.

Examples

```
library(nprcgenekeepr)
data("qcBreeders")
data("pedWithGenotype")
available <- c(
  "JGPN6K", "8KM1MP", "I9TQ0T", "Q0RGP7", "VFS0XB", "CQC133",
  "2KULR3", "HOYW0S", "FHV13N", "OUM6QF", "6Z7MD9", "CFPEEU",
  "HLI95R", "RI0O7F", "7M51X5", "DR5GXB", "170ZTZ", "C1ICXL"
)
nonMales <- c(
  "JGPN6K", "8KM1MP", "I9TQ0T", "Q0RGP7", "CQC133",
  "2KULR3", "HOYW0S", "FHV13N", "OUM6QF", "6Z7MD9", "CFPEEU",
  "HLI95R", "RI0O7F", "7M51X5", "DR5GXB", "170ZTZ", "C1ICXL"
)
male <- "VFS0XB"
calculateSexRatio(ids = male, ped = pedWithGenotype)
calculateSexRatio(ids = nonMales, ped = pedWithGenotype)
calculateSexRatio(ids = available, ped = pedWithGenotype)
calculateSexRatio(
  ids = available, ped = pedWithGenotype,
  additionalMales = 1L
)
calculateSexRatio(
  ids = available, ped = pedWithGenotype,
  additionalFemales = 1L
)
calculateSexRatio(
  ids = available, ped = pedWithGenotype,
  additionalMales = 1, additionalFemales = 1L
)
calculateSexRatio(
  ids = nonMales, ped = pedWithGenotype,
  additionalMales = 1, additionalFemales = 0L
)
calculateSexRatio(
  ids = character(0), ped = pedWithGenotype,
  additionalMales = 1, additionalFemales = 0L
)
```

---

checkAncestryRules *Validate an ancestry compatibility rules table*

Description

Checks the structure and domain of a center-configurable ancestry compatibility rules table (issue #168). Each row is one unordered pair of standardized ancestry levels plus a severity: `block` rules exclude the pairing during breeding-group formation, `flag` rules annotate it afterward. It mirrors `checkKinshipOverrides`: it `stop()`s on structural or domain errors and returns the coerced table when the input is acceptable. An empty table (zero rules) is valid.

Usage

```
checkAncestryRules(rules)
```

Arguments

rules	data.frame with columns ancestry1 and ancestry2 (standardized ancestry levels) and severity ("block" or "flag"); each row is one unordered level pair. Any extra columns are kept in the returned table.

Details

Ancestry levels must come from convertAncestry's standardized vocabulary: CHINESE, INDIAN, HYBRID, JAPANESE, OTHER, UNKNOWN. Levels are coerced to uppercase and severity to lowercase before validation, so a hand-edited file's casing never matters. A rule may pair a level with itself (e.g. HYBRID with HYBRID); duplicated unordered pairs are a data error the user must resolve. Because convertAncestry maps a blank ancestry to UNKNOWN but any unrecognized text – including a literal re-standardized "UNKNOWN" string – to OTHER, a table that names one of UNKNOWN/OTHER without the other draws a warning here: a center wanting conservative treatment of animals without usable ancestry information almost always wants both.

Value

The validated rules data.frame with ancestry1 and ancestry2 coerced to uppercase character and severity to lowercase character. Leading and trailing white space is trimmed from these three columns.

Examples

```
rules <- data.frame(
  ancestry1 = c("INDIAN", "INDIAN"),
  ancestry2 = c("CHINESE", "HYBRID"),
  severity = c("block", "flag"), stringsAsFactors = FALSE
)
checkAncestryRules(rules)
```

---

checkChangedColsLst *Check a changed-columns list for non-empty fields*

Description

Check a changed-columns list for non-empty fields

Usage

```
checkChangedColsLst(changedCols)
```

Arguments

changedCols	list with fields for each type of column change made by qcStudbook.

Value

Returns TRUE if any changed-columns field is non-empty, otherwise FALSE.

Examples

```
library(nprcgenekeepr)
library(lubridate)
pedOne <- data.frame(
  ego_id = c(
    "s1", "d1", "s2", "d2", "o1", "o2", "o3",
    "o4"
  ),
  `si re` = c(NA, NA, NA, NA, "s1", "s1", "s2", "s2"),
  dam_id = c(NA, NA, NA, NA, "d1", "d2", "d2", "d2"),
  sex = c("F", "M", "M", "F", "F", "F", "F", "M"),
  birth_date = mdy(
    paste0(
      sample(1:12, 8, replace = TRUE), "-",
      sample(1:28, 8, replace = TRUE), "-",
      sample(seq(0, 15, by = 3), 8, replace = TRUE) +
        2000
    )
  ),
  stringsAsFactors = FALSE, check.names = FALSE
)

errorLst <- qcStudbook(pedOne, reportErrors = TRUE, reportChanges = TRUE)
checkChangedColsLst(errorLst$changedCols)
```

---

checkCrossCenterMapping

Collect every cross-center identity-mapping problem, without stopping

---

Description

The "show every problem at once" companion to resolveCrossCenterIds (issue #149 Slice 1), sharing its five validation checks – id existence, mapping uniqueness, undeclared id collisions, a pedB animal that reuses an id the mapping gives a pedA animal, and conflicting recorded parents – via the same internal helpers (D2, docs/planning/issue149-cross-center-identity-mapping-workflow-plan.md section 3). resolveCrossCenterIds() stop()s on the first problem it finds; checkCrossCenterMapping() never stop()s on a domain problem – every one found becomes a row in the returned data.frame instead, so a curator can see and fix every issue at once rather than one at a time. A structural problem (a required column missing from any of the three inputs) still stop()s immediately, matching every other checkXxx() function in this package (checkKinshipOverrides, checkTwinRelations).

Usage

```
checkCrossCenterMapping(pedA, pedB, mapping)
```

Arguments

pedA	a pedigree data.frame for the first center, with (at least) columns id, sire, and dam.
pedB	a pedigree data.frame for the second center, with (at least) columns id, sire, and dam.
mapping	a data.frame with columns idA and idB: one row per curator-proposed cross-center identity link.

Details

Existence and uniqueness problems (tier A) are checked first; if either is present, only those are returned and the collision, reused-id and conflict checks (tier B) are skipped entirely, since a mapped id that does not resolve to a real pedigree row makes those checks meaningless. Tier B – and all three of its checks, across every mapped pair – runs only once tier A is clean.

Value

A data.frame of every domain problem found, with columns type ("existence", "uniqueness", "collision", "mappedIdClash", or "conflict"), ids (the offending id(s), as a single comma-separated string), and message (a human-readable description). Zero rows means none of these five kinds of problem was found. A "mappedIdClash" row names an id of a pedB animal with no mapping row that the mapping already uses for a pedA animal; a merge would hold two rows with that id. resolveCrossCenterIds can still stop on conflicting values in other columns shared by both pedigrees (e.g. sex), which this function does not check.

See Also

resolveCrossCenterIds

Examples

```
library(nprcgenekeepr)
pedA <- data.frame(
  id = c("P1", "P2", "T1"), sire = c(NA, NA, "P1"), dam = c(NA, NA, "P2"),
  stringsAsFactors = FALSE
)
pedB <- data.frame(
  id = c("X9", "O1"), sire = c(NA, "X9"), dam = c(NA, NA),
  stringsAsFactors = FALSE
)
mapping <- data.frame(idA = "T1", idB = "X9", stringsAsFactors = FALSE)
checkCrossCenterMapping(pedA, pedB, mapping) # zero rows: clean
```

checkErrorLst *Check an error list for non-empty fields*

Description

Check an error list for non-empty fields

Usage

```
checkErrorLst(errorLst)
```

Arguments

errorLst	list with fields for each type of error detectable by qcStudbook.

Value

Returns FALSE if all fields are empty or the list is NULL otherwise TRUE. The changedCols field is not examined; use checkChangedColsLst for it.

Examples

```
errorLst <- qcStudbook(nprcgenekeepr::pedFemaleSireMaleDam,
  reportErrors = TRUE
)
checkErrorLst(errorLst)
```

checkGenotypeFile *Check genotype file*

Description

Checks to ensure the content and structure are appropriate for a genotype file. These checks are simply based on expected columns and legal domains. The function stops (an error, not a warning) when the dataframe has fewer than three columns, when the first column name does not contain "id" (any case), when any column is named first or second (any case), or when an allele in columns 2 or 3 that reads as an integer is above 10000 (it would collide with the integer codes addGenotype assigns).

Usage

```
checkGenotypeFile(genotype)
```

Arguments

genotype	dataframe with genotype data

Value

The genotype dataframe, checked for the column count, the first column's name and the allele range described above; no column types are checked. The returned genotype file has the first column name forced to "id".

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped <- ped[order(ped$id), ]
genotype <- data.frame(
  id = ped$id[50 + 1:20],
  first_name = paste0("first_name", 1:20),
  second_name = paste0("second_name", 1:20),
  stringsAsFactors = FALSE
)

## checkGenotypeFile disallows dataframe with < 3 columns
tryCatch(
  {
    checkGenotypeFile(genotype[, c("id", "first_name")])
  },
  warning = function(w) {
    cat("Warning produced")
  },
  error = function(e) {
    cat("Error produced")
  }
)
```

---

checkKinshipOverrides *Validate a kinship overrides table*

Description

Checks the structure and domain of an outside-information kinship override table. The table supplies pairwise kinship coefficients (`id1`, `id2`, `kinship`) that `applyKinshipOverrides` writes into a computed kinship matrix, replacing the pedigree-derived value for those pairs. It mirrors `checkGenotypeFile`: it `stop()`s on structural or domain errors and returns the (id-coerced) table when the input is acceptable.

Usage

```
checkKinshipOverrides(overrides)
```

Arguments

`overrides`	data.frame with id columns `id1` and `id2` and a numeric `kinship` column; each row is one off-diagonal pair. Any extra columns are ignored.

Details

kinship is the kinship coefficient f (the probability that an allele drawn at random from each of the two animals is identical by descent), **not** the coefficient of relatedness r (= $2f$ for non-inbred animals). Supplying r – e.g. 0.25 for half-sibs whose true f is 0.125 – silently corrupts the matrix, so an off-diagonal value above 0.5 (the maximum for a non-inbred pair) draws a warning here. It also stop()s on a missing (NA) or negative kinship, on a row whose id1 equals its id2, and on a duplicated unordered pair. The exact positive-semi-definiteness bound is enforced by applyKinshipOverrides once the matrix diagonal is known.

Value

The validated overrides data.frame with id1 and id2 coerced to character.

Examples

```
overrides <- data.frame(
  id1 = c("A1", "A3"), id2 = c("A2", "A4"),
  kinship = c(0.25, 0.125), stringsAsFactors = FALSE
)
checkKinshipOverrides(overrides)
```

---

checkLinkageMarkerGenotypeFile

Check a long-format multi-locus marker genotype file, multiallelic-tolerant

---

Description

Validates the structure of a long-format marker genotype table (one row per id x locus), the same schema checkMarkerGenotypeFile checks – but, unlike that function, does not require every locus to be biallelic. This is a new, sibling validator for the linkage-aware and haplotype-block metrics family (issue #153): real colony marker panels (e.g. microsatellite/STR panels) are routinely multiallelic, a data shape the KING-robust kinship estimator checked by checkMarkerGenotypeFile cannot represent, but which buildMarkerGenotypeMatrix pivots without error. checkMarkerGenotypeFile itself, and everything downstream of it (markerKinship), is untouched by this function.

Usage

```
checkLinkageMarkerGenotypeFile(genotype)
```

Arguments

genotype	dataframe with long-format marker genotype data: exactly four columns, id, locus, allele1, allele2 (one row per individual x locus).

Details

All of checkMarkerGenotypeFile's structural checks are retained – exactly four columns, id as the first column, no duplicate id x locus rows – except the per-locus more-than-two-distinct-alleles rejection, which is deliberately omitted.

Value

The genotype dataframe, checked to ensure the column count, first-column identity, and row uniqueness are all valid. The returned dataframe has its column names forced to c("id", "locus", "allele1","allele2").

See Also

checkMarkerGenotypeFile, buildMarkerGenotypeMatrix

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("W", "X", "Y", "Z"),
  locus = c("L1", "L1", "L1", "L1"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("B", "C", "A", "D"),
  stringsAsFactors = FALSE
)
checkLinkageMarkerGenotypeFile(markerGenotype)
```

---

checkLocusMetadata *Check a locus-metadata sidecar table and classify per-locus coverage*

Description

Validates the structure of a locus-metadata sidecar table (locus, chrom, pos, and optionally cM) – the schema introduced by issue #152's own design decision and reused verbatim by issue #153 – and classifies each locus into one of three coverage tiers, following a PLINK-style three-state coverage model: "full" (both chrom and pos present; cM is optional even within "full"), "partial" (exactly one of chrom/pos present), or "none" (neither present). A value counts as present when it is not NA; an empty or all-space chrom is treated as missing and is returned as NA.

Usage

```
checkLocusMetadata(locusMetadata)
```

Arguments

locusMetadata	dataframe with locus metadata: three or four columns, locus, chrom, pos, and optionally cM (one row per locus).

Details

A locus-metadata table has one row per locus (not per individual x locus, unlike checkMarkerGenotypeFile's genotype table). Real curated marker panels typically supply little to no locus-order metadata – this function reports coverage explicitly per locus rather than requiring complete metadata before any downstream use.

Value

The locus-metadata dataframe, checked to ensure the column count, first-column identity, and row uniqueness are all valid, with a new `coverage` column appended (`"full"`/`"partial"`/ `"none"`). The returned dataframe has `locus` and `chrom` coerced to character, and its column names forced to `locus`, `chrom`, `pos` (and `cM` when there are four columns) whatever the input names were.

References

Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M. A. R., Bender, D., Maller, J., Sklar, P., de Bakker, P. I. W., Daly, M. J., & Sham, P. C. (2007). PLINK: a tool set for whole-genome association and population-based linkage analyses. *American Journal of Human Genetics*, 81(3), 559-575. doi:10.1086/519795

See Also

checkMarkerGenotypeFile

Examples

```
library(nprcgenekeepr)
locusMetadata <- data.frame(
  locus = c("L1", "L2", "L3"),
  chrom = c("1", "1", NA),
  pos = c(1000000, NA, NA),
  stringsAsFactors = FALSE
)
checkLocusMetadata(locusMetadata)
```

---

checkMarkerGenotypeFile

Check a long-format multi-locus marker genotype file

Description

Validates the structure and legal domain of a long-format marker genotype table (one row per `id` x `locus`), the input format for the marker-based (KING-robust) kinship estimator (markerKinship). This is a new, sibling schema to the single-locus `first_name`/`second_name` genotype format checked by checkGenotypeFile – that function, and everything downstream of it (addGenotype, geneDrop), is untouched by this one.

Usage

```
checkMarkerGenotypeFile(genotype)
```

Arguments

genotype	dataframe with long-format marker genotype data: exactly four columns, `id`, `locus`, `allele1`, `allele2` (one row per individual x locus).

Details

The KING-robust kinship estimator (Manichaikul et al. 2010) is defined for biallelic markers only – every genotype is classified as homozygous-reference, heterozygous, or homozygous-alternate, with no representation for a third allele at a locus. A locus with more than two distinct alleles observed anywhere in the input is therefore rejected outright, rather than silently producing an uninterpretable kinship estimate.

Value

The genotype dataframe, checked to ensure the column count, first-column identity, per-locus allele count, and row uniqueness are all valid. The returned dataframe has its column names forced to `c("id", "locus", "allele1", "allele2")`.

References

Manichaikul, A., Mychaleckyj, J. C., Rich, S. S., Daly, K., Sale, M., & Chen, W.-M. (2010). Robust relationship inference in genome-wide association studies. *Bioinformatics*, 26(22), 2867-2873. doi:10.1093/bioinformatics/btq559

See Also

`buildMarkerGenotypeMatrix`, `markerKinship`

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("A", "A", "B", "B"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("A", "B", "B", "B"),
  stringsAsFactors = FALSE
)
checkMarkerGenotypeFile(markerGenotype)
```

checkMhcHaplotypeFile *Check a wide-format MHC haplotype designation file*

Description

Validates the structure of a wide per-animal MHC haplotype designation table: exactly three columns – `id` plus the animal's two named haplotype designations, one row per animal. This is the explicit input designation for MHC haplotype reporting (issue #148): a curator designates data as MHC by supplying it through this format, and no MHC semantics are ever inferred from locus or column names (the format has no locus names). A new, sibling input family: the long-format marker validators (`checkMarkerGenotypeFile`, `checkLinkageMarkerGenotypeFile`, `checkSequenceGenotypeFile`) and the single-locus `checkGenotypeFile` path are untouched by this one.

Usage

```
checkMhcHaplotypeFile(genotype)
```

Arguments

`genotype`	dataframe with wide-format MHC haplotype data: exactly three columns, `id`, `haplotype1`, `haplotype2` (one row per animal; any column names, see Details).

Details

Column names are not prescribed: the first column must be id-like (case-insensitive match on `"id"`) and is forced to `id`; the second and third are forced to `haplotype1` and `haplotype2`, so a colony file with, e.g., `first_name`/`second_name` headers (the bundled `rhesusGenotypes` shape) loads unchanged. `NA`/empty cells pass validation – a missing call is the statistics layer's concern, not a structural error. A trailing `?` on a designation (an uncertain, provisional call) is likewise not validation's business; see the issue #148 design plan's D3 parse rule. Haplotype designations are otherwise opaque labels: never parsed, split, or matched against MHC region names.

Value

The genotype dataframe, checked to ensure the column count, first-column identity, and row uniqueness are all valid. The returned dataframe has its column names forced to `c("id", "haplotype1","haplotype2")`.

See Also

`rhesusGenotypes`, `checkMarkerGenotypeFile`

Examples

```
library(nprcgenekeepr)
checked <- checkMhcHaplotypeFile(rhesusGenotypes)
head(checked)
```

checkParentAge *Check parent ages against a minimum age*

Description

Ensure parents are sufficiently older than offspring

Usage

```
checkParentAge(
  sb,
  minSireAge = NULL,
  minDamAge = NULL,
  minParentAge = lifecycle::deprecated(),
  reportErrors = FALSE
)
```

Arguments

sb	A dataframe containing a table of pedigree and demographic information. It needs the columns id, sire, dam, birth and exit; species is optional (see minSireAge). Without exit the function fails with an uninformative error.
minSireAge	numeric minimum age in years for a male to have sired an offspring. NULL (default) looks up the floor for each sire's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor for all sires. The check is not performed for animals with missing birth dates.
minDamAge	numeric minimum age in years for a female to have borne an offspring. NULL (default) looks up the floor for each dam's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor for all dams.
minParentAge	**[Deprecated]** Deprecated scalar minimum parent age. Supplying it sets both minSireAge and minDamAge; use those sex-specific parameters instead.
reportErrors	logical value. If TRUE, NULL is returned instead of stopping (or returning sb) when the input cannot be checked: the data frame has no rows, lacks id, sire or dam, or birth is not of class Date, POSIXct or character. It does not change the result for valid input.

Value

A dataframe containing rows for each animal where one or more parent was younger than the minimum breeding age (minSireAge or minDamAge, or the species floor when they are NULL) at the animal's birth. It contains all of the columns in the original sb dataframe, with exit converted to character and the rows reordered, plus the following added columns:

1. sireBirth – sire's birth date

2. sireAge – age of sire in years on the date indicated by birth.
3. damBirth – dam's birth date
4. damAge – age of dam in years on the date indicated by birth.

Examples

```
library(nprcgenekeepr)
qcPed <- nprcgenekeepr::qcPed
checkParentAge(qcPed, minSireAge = 2L, minDamAge = 2L)
checkParentAge(qcPed, minSireAge = 3L, minDamAge = 3L)
checkParentAge(qcPed, minSireAge = 5L, minDamAge = 5L)
checkParentAge(qcPed, minSireAge = 6L, minDamAge = 6L)
head(checkParentAge(qcPed, minSireAge = 10L, minDamAge = 10L))
```

---

checkRequiredCols *Check column names for required columns*

Description

Check column names for required columns

Usage

```
checkRequiredCols(cols, reportErrors)
```

Arguments

cols	character vector of column names
reportErrors	logical value with no default. When TRUE and required columns are missing, a character vector of the names of the missing columns is returned. When FALSE and required columns are missing, the program stops with the error "Required field(s) missing: ...".

Details

When reportErrors = TRUE, NA entries in cols are treated as ordinary non-matching column names when building the list of missing required columns, rather than causing an error. (Earlier versions could error with "missing value where TRUE/FALSE needed" on such out-of-contract input.)

Value

NULL is returned if all required columns are present. See description of reportErrors for what happens when required columns are missing.

Examples

```
library(nprcgenekeepr)
requiredCols <- getRequiredCols()
cols <- strsplit(
  paste0(
    "id,sire,siretype,dam,damtype,sex,numberofparentsknown,birth,",
    "arrivalatcenter,death,departure,status,ancestry,fromcenter?,",
    "origin"
  ),
  ","
)[[1L]]
checkRequiredCols(cols, reportErrors = TRUE) # NULL: all required present
# A missing required column is returned by name
checkRequiredCols(setdiff(cols, "birth"), reportErrors = TRUE)
```

---

checkSequenceGenotypeFile

Check a long-format sequence-derived marker genotype file

Description

Validates the structure of a long-format marker genotype table (one row per `id` x `locus`), the same schema `checkMarkerGenotypeFile` checks – but sized and hardened for sequence-derived panels (issue #152): a soft, overridable warning above a sparse/GBS-scale panel-size ceiling, and an explicit rejection of a literal `"."` allele value (VCF's missing-genotype placeholder), rather than silently counting it as a real allele. Optionally validates the accompanying locus-metadata sidecar's own structure by reusing `checkLocusMetadata`, rather than reinventing that check; the sidecar is not cross-checked against the loci of the genotype table.

Usage

```
checkSequenceGenotypeFile(genotype, locusMetadata = NULL, maxLoci = 50000L)
```

Arguments

`genotype`	dataframe with long-format marker genotype data: exactly four columns, `id`, `locus`, `allele1`, `allele2` (one row per individual x locus).
`locusMetadata`	optional dataframe with locus metadata (see `checkLocusMetadata`): `locus`, `chrom`, `pos`, and optionally `cM`, one row per locus. When supplied, is validated via `checkLocusMetadata` – its own violations propagate unchanged. Defaults to `NULL` (no sidecar to validate).
`maxLoci`	numeric scope-tier ceiling (default `50000L`, this package's own sparse/GBS-scale target ceiling) above which a locus count triggers a `warning()` rather than a `stop()`.

Details

All of checkMarkerGenotypeFile's structural checks are retained – exactly four columns, id as the first column, no duplicate id x locus rows, and the same biallelic-only rejection the KING-robust kinship estimator (Manichaikul et al. 2010) requires. Two rules are new: a literal "." allele value is rejected before the biallelic count is even checked, so a curator sees the correct, actionable error rather than a misleading "more than two alleles" report (a genotype-preprocessing pipeline emitting VCF-style missingness that was never converted to NA is a real, anticipated failure mode – see Danecek et al. 2011 for the VCF missing-genotype convention this guards against); and a locus count above maxLoci produces a warning(), not a stop() – the "right" ceiling for this package's own vectorized implementations is not yet empirically known, so a hard stop would risk blocking legitimate data.

Value

The genotype dataframe, checked to ensure the column count, first-column identity, per-locus allele count, absence of a literal "." placeholder, and row uniqueness are all valid. The returned dataframe has its column names forced to c("id", "locus", "allele1","allele2").

References

Manichaikul, A., Mychaleckyj, J. C., Rich, S. S., Daly, K., Sale, M., & Chen, W.-M. (2010). Robust relationship inference in genome-wide association studies. *Bioinformatics*, 26(22), 2867-2873. doi:10.1093/bioinformatics/btq559

Danecek, P., et al. (2011). The variant call format and VCFtools. *Bioinformatics*, 27(15), 2156-2158. doi:10.1093/bioinformatics/btr330

See Also

checkMarkerGenotypeFile, checkLinkageMarkerGenotypeFile, checkLocusMetadata, buildMarkerGenotypeMatrix

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("A", "A", "B", "B"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("A", "B", "B", "B"),
  stringsAsFactors = FALSE
)
checkSequenceGenotypeFile(markerGenotype)
```

`checkSnapshotHistory` *Validate a colony snapshot-history table*

Description

Checks the structure and domain of a longitudinal colony snapshot history (issue #167): one row per recorded snapshot, in the 27-column version-1 schema. It mirrors `checkKinshipOverrides`: it `stop()`s with a specific message on structural or domain errors and returns the coerced history when the input is acceptable.

Usage

```
checkSnapshotHistory(history)
```

Arguments

`history`	data.frame holding the snapshot history, typically from `readSnapshotHistory`; one row per snapshot.

Details

The schema has three groups of nine columns:

Provenance / comparability `schemaVersion`, `snapshotDate`, `packageVersion`, `membershipRule`, `guIter`, `guThresh`, `nAnimals`, `nMales`, `nFemales`. These let successive snapshots be compared like with like: snapshots generated under different membership rules, `guIter` settings, or package versions are comparable only with care, and trend displays flag such mixed-provenance series rather than hiding them.

Colony scalars `fe`, `fg`, `fgSE`, `neGD`, `neSexRatio`, `neVariance`, `nMaleFounders`, `nFemaleFounders`, `nFounders` — verbatim from `reportGV`.

Colony aggregates `meanIndivMeanKin`, `medianIndivMeanKin`, `skewnessIndivMeanKin`, `kurtosisIndivMeanKin`, `meanGu`, `medianGu`, `meanGuSE`, `skewnessGu`, `kurtosisGu` — the Summary Statistics definitions applied to the per-animal `indivMeanKin` and gu report columns.

Violations rejected: missing columns, non-numeric metric or count fields, an unrecognized `schemaVersion`, a duplicated (`snapshotDate`, `membershipRule`) pair, and malformed (non ISO-8601) dates. The `membershipRule` field is an open string — no enumeration is enforced, so new rule names are additive. Extra columns are ignored, matching the sibling validators.

Value

The validated history with `snapshotDate` coerced to `Date`, `packageVersion` and `membershipRule` coerced to character, and `schemaVersion` and the count columns coerced to integer.

Examples

```
history <- readSnapshotHistory(system.file("extdata", "examples",
  "example_snapshot_history.csv",
  package = "nprcgenekeepr"
))
history <- checkSnapshotHistory(history)
```

`checkTwinRelations` *Validate a twin/zygosity relations table*

Description

Checks the structure and domain of a twin/zygosity sidecar table. The table supplies pairwise twin declarations (`id1`, `id2`, `code`) that record which individuals in a pedigree are twins and with what twin zygosity certainty – a fact this package's per-individual pedigree data frame cannot represent directly. It mirrors `checkKinshipOverrides`: it `stop()`s on structural or domain errors and returns the (id-coerced) table when the input is acceptable.

Usage

```
checkTwinRelations(ped, twinRelations)
```

Arguments

`ped`	a pedigree data.frame with (at least) `id`, `sire`, `dam`, and `sex` columns.
`twinRelations`	data.frame with columns `id1`, `id2`, `code`; each row is one twin declaration. Any extra columns are ignored.

Details

`code` adopts kinship2's own literal twin-code labels – `"MZ twin"`, `"DZ twin"`, `"UZ twin"` – rather than a bare `"zygosity"` identifier, which would collide with this package's unrelated marker-genetics heterozygosity vocabulary (see `markerObservedHeterozygosity`). kinship2's own fourth, non-twin `"spouse"` code is out of scope here.

Validation reproduces kinship2's own `relation` acceptance rules, confirmed empirically against the installed kinship2 namespace: both ids must exist in `ped` and differ (all codes); an `"MZ twin"`/`"DZ twin"` pair must already share both `sire` and `dam` in `ped`; an `"MZ twin"` pair must additionally have matching `sex`; `"UZ twin"` has no such precondition.

Value

The validated `twinRelations` data.frame with `id1` and `id2` coerced to character.

Examples

```
ped <- data.frame(
  id = c("F1", "F2", "S1", "S2"),
  sire = c(NA, NA, "F1", "F1"),
  dam = c(NA, NA, "F2", "F2"),
  sex = c("M", "F", "F", "F"),
  stringsAsFactors = FALSE
)
twinRelations <- data.frame(
  id1 = "S1", id2 = "S2", code = "MZ twin", stringsAsFactors = FALSE
)
checkTwinRelations(ped, twinRelations)
```

---

chooseAlleles *Combine two allele vectors by Mendelian sampling*

Description

Combine two allele vectors by Mendelian sampling

Usage

```
chooseAlleles(a1, a2)
```

Arguments

a1	integer vector with the alleles of one parent, one per simulated iteration
a2	integer vector with the other alleles of the same parent, one per simulated iteration. a1 and a2 are expected to be equal length vectors; if they are not, the shorter is recycled silently.

Value

An integer vector with the result of sampling from a1 and a2 according to Mendelian inheritance.

Examples

```
chooseAlleles(0L:4L, 5L:9L)
```

chooseDate *Choose the earlier or later of two dates*

Description

Part of Pedigree Curation

Usage

```
chooseDate(d1, d2, earlier = TRUE)
```

Arguments

d1	single Date value, the first of two dates to compare.
d2	single Date value, the second of two dates to compare.
earlier	logical variable with TRUE if the earlier of the two dates is to be returned, otherwise the later is returned. Default is TRUE.

Details

Given two dates, one is selected to be returned based on whether it occurred earlier or later than the other. NAs are ignored if possible.

Value

A single Date value, the chosen date, or NA where neither is provided. Vectors longer than one are not supported; use pmin or pmax, or mapply, for vectors.

Examples

```
library(nprcgenekeepr)
someDates <- lubridate::mdy(paste0(
  sample(1:12, 2, replace = TRUE), "-",
  sample(1:28, 2, replace = TRUE), "-",
  sample(seq(0, 15, by = 3), 2,
    replace = TRUE
  ) + 2000
))
someDates
chooseDate(someDates[1], someDates[2], earlier = TRUE)
chooseDate(someDates[1], someDates[2], earlier = FALSE)
```

computeGenomicROH *Compute genomic Runs of Homozygosity (ROH) and F_ROH*

Description

Computes per-individual genomic Runs of Homozygosity (ROH) segments and the genomic inbreeding coefficient F_ROH from a sequence-scale marker genotype matrix (issue #152 Slice 3, design decision D6) – a sequence-verified inbreeding estimate independent of (often incomplete) pedigree records, complementing rather than duplicating this package's existing pedigree-based kinship/founder metrics.

Usage

```
computeGenomicROH(genotypeMatrix, locusMetadata, minSnp = 50L, minBp = 1e+06)
```

Arguments

genotypeMatrix	a character matrix as returned by buildMarkerGenotypeMatrix: rows are individual ids, columns are loci, and each cell is that individual's two alleles at that locus, sorted and joined by "/" (e.g. "A/B"), or NA if not genotyped at that locus.
locusMetadata	dataframe with locus metadata (see checkLocusMetadata): locus, chrom, pos, and optionally cM, one row per locus. Required – absent or NULL triggers an early stop() rather than a silent NA.
minSnp	integer minimum number of consecutive homozygous loci required for a run to qualify as an ROH segment. Default 50L, scaled down from PLINK's own literal default (100) for this package's sparser sparse/GBS-scale target tier (design doc D1).
minBp	numeric minimum base-pair span (last locus pos minus first locus pos) required for a run to qualify as an ROH segment. Default 1e6 (1 Mb), matching PLINK's own --homozyg-kb default and Ceballos et al. (2018)'s commonly-cited ROH-calling threshold.

Details

Within each chromosome, loci are ordered by pos. A "run" is a maximal, gapless stretch of homozygous, non-missing genotyped loci – both a heterozygous call and a missing (NA) call end the current run. A run qualifies as an ROH segment only if it has at least minSnp loci AND spans at least minBp base pairs (the field-standard dual threshold, matching PLINK's --homozyg-snp/ --homozyg-kb, Purcell et al. 2007) – either condition alone is not sufficient. totalRohLength sums the qualifying segments' spans across all chromosomes; fRoh divides that sum by genomeLength, a single value shared across every individual: the sum, per chromosome, of (max(pos) - min(pos)) among the full-coverage loci in locusMetadata that are also columns of genotypeMatrix (Ceballos et al. 2018's $L_{autosome}$ convention). The denominator is fixed across individuals within one call, which avoids conflating an individual's own missingness with inbreeding, but it changes with the matrix's locus set: full-coverage loci of locusMetadata that are not columns of genotypeMatrix are left

out of that length, with one warning naming them (at most 10 are named, then "and N more"), so `fRoh` is comparable across a cohort only when the same loci are used. A locus without a position never enters the length, so its absence is not reported.

Only loci with full `locusMetadata` coverage (both `chrom` and `pos` present, per `checkLocusMetadata`'s own three-tier classification) can be placed in the ordered walk; a locus lacking full coverage, or entirely absent from `locusMetadata`, is excluded from both the walk and the `genomeLength` denominator, with a warning naming it – it is never treated as a run-breaking gap. When every chromosome has at most one full-coverage locus, `genomeLength` is 0 and `fRoh` is an undefined `0/0` ratio: `NA` with a warning, not `NaN` silently, mirroring `markerFst`'s own undefined-ratio precedent.

Value

A per-individual dataframe: `id`, `nSegments` (count of qualifying ROH segments), `totalRohLength` (their summed bp span), and `fRoh` (the genomic inbreeding coefficient, or `NA` if `genomeLength` is 0).

References

Ceballos, F. C., Joshi, P. K., Clark, D. W., Ramsay, M., & Wilson, J. F. (2018). Runs of homozygosity: windows into population history and trait architecture. *Nature Reviews Genetics*, 19(4), 220-234. doi:10.1038/nrg.2017.109

Purcell, S., et al. (2007). PLINK: a tool set for whole-genome association and population-based linkage analyses. *American Journal of Human Genetics*, 81(3), 559-575. doi:10.1086/519795

See Also

`buildMarkerGenotypeMatrix`, `checkLocusMetadata`, `checkSequenceGenotypeFile`

Examples

```
library(nprcgenekeepr)
locusMetadata <- data.frame(
  locus = c("L1", "L2", "L3", "L4"),
  chrom = c("1", "1", "1", "1"),
  pos = c(0, 400000, 900000, 1400000),
  stringsAsFactors = FALSE
)
genotypeMatrix <- matrix(
  c("A/A", "A/A", "A/B", "A/A"),
  nrow = 1L, dimnames = list("Animal1", locusMetadata$locus)
)
computeGenomicROH(genotypeMatrix, locusMetadata, minSnp = 2L, minBp = 300000)
```

convertAncestry *Convert ancestry information to a standard code*

Description

Part of Pedigree Curation

Usage

```
convertAncestry(ancestry)
```

Arguments

ancestry	character vector or NA with free-form text providing information about the geographic population of origin. Matching ignores case and looks for text inside each value: a value containing "chin" but not "ind" is CHINESE, one containing "ind" but not "chin" is INDIAN, one containing both, or "hyb", is HYBRID, and one containing "jap" is JAPANESE. NA is UNKNOWN and anything else is OTHER. For example, "Indonesian" is INDIAN.

Value

A factor vector of standardized designators specifying if an animal is a Chinese rhesus, Indian rhesus, Chinese-Indian hybrid rhesus, or Japanese macaque. Levels: CHINESE, INDIAN, HYBRID, JAPANESE, OTHER, UNKNOWN.

Examples

```
original <- c("china", "india", "hybridized", NA, "human", "gorilla")
convertAncestry(original)
```

convertDate *Convert character date columns to Date type*

Description

Part of Pedigree Curation

Usage

```
convertDate(ped, timeOrigin = as.Date("1970-01-01"), reportErrors = FALSE)
```

Arguments

ped	a dataframe of pedigree information that may contain birth, death, departure, or exit dates. The fields are optional, but will be used if present.(optional fields: birth, death, departure, and exit). If it has a recordStatus column, only the records marked "added" (animals added as stand-ins for unknown parents) are left unconverted. Every other record, including one whose recordStatus is NA or not recognized, is a real animal and has its dates converted and checked.
timeOrigin	date object used by as.Date to set origin.
reportErrors	logical value. If FALSE (default), an invalid date stops with an error naming the column and the rows. If TRUE, every date column is scanned and the converted pedigree is not returned; instead the result is a sorted character vector of the row numbers with an invalid date (a row is repeated once for each column in which it has one), or NULL when there are none. Row numbers, here and in the error message, are positions in ped as passed in, so records marked "added" count toward the numbering wherever they sit.

Value

A dataframe with an updated table with date columns converted from character data type to Date data type. A date must be in the form YYYY-MM-DD (month and day may have one digit), or YYYYMMDD when no date in the column has a separator. A year before 1000 is invalid. A value that does not conform is an error (see reportErrors), not set to NA. NA values and blank strings are left as NA. A date column must be character, factor, logical, integer or Date; any other class stops with an error. Records marked "added" are not checked and are returned after the other records. With reportErrors = TRUE see that argument instead.

Examples

```
library(lubridate)
set_seed(10)
someBirthDates <- paste0(
  sample(seq(0, 15, by = 3), 10,
    replace = TRUE
  ) + 2000, "-",
  sample(1:12, 10, replace = TRUE), "-",
  sample(1:28, 10, replace = TRUE)
)
someBadBirthDates <- paste0(
  sample(1:12, 10, replace = TRUE), "-",
  sample(1:28, 10, replace = TRUE), "-",
  sample(seq(0, 15, by = 3), 10,
    replace = TRUE
  ) + 2000
)
someDeathDates <- sample(someBirthDates, length(someBirthDates),
  replace = FALSE
)
someDepartureDates <- sample(someBirthDates, length(someBirthDates),
  replace = FALSE
)
```

```
ped1 <- data.frame(
  birth = someBadBirthDates, death = someDeathDates,
  departure = someDepartureDates
)
someDates <- ymd(someBirthDates)
ped2 <- data.frame(
  birth = someDates, death = someDeathDates,
  departure = someDepartureDates
)
ped3 <- data.frame(
  birth = someBirthDates, death = someDeathDates,
  departure = someDepartureDates
)
someNADeathDates <- someDeathDates
someNADeathDates[c(1, 3, 5)] <- ""
someNABirthDates <- someDates
someNABirthDates[c(2, 4, 6)] <- NA
ped4 <- data.frame(
  birth = someNABirthDates, death = someNADeathDates,
  departure = someDepartureDates
)

## convertDate identifies bad dates
result <- tryCatch(
  {
    convertDate(ped1)
  },
  warning = function(w) {
    print("Warning in date")
  },
  error = function(e) {
    print("Error in date")
  }
)

## convertDate with error flag returns error list and not an error
convertDate(ped1, reportErrors = TRUE)

## convertDate recognizes good dates
all(is.Date(convertDate(ped2)$birth))
all(is.Date(convertDate(ped3)$birth))

## convertDate handles NA and empty character string values correctly
convertDate(ped4)
```

`convertFromCenter` *Convert from-center information to a logical value*

Description

Part of Pedigree Curation

Usage

```
convertFromCenter(fromCenter)
```

Arguments

fromCenter	character or logical vector or NA indicating whether or not the animal is from the center.

Value

A logical vector specifying TRUE if an animal is from the center otherwise FALSE. An NA input stays NA. A value is read only when the whole text, ignoring case and any spaces around it, is y, yes, t or true (TRUE), or n, no, f or false (FALSE). Any other value, such as "unknown", "maybe", a number such as 1L or an empty string, stops with an error naming the offending rows.

Examples

```
original <- c(
  "y", "yes", "Y", "Yes", "YES", "n", "N", "No", "NO", "no",
  "t", "T", "True", "true", "TRUE", "f", "F", "false", "False",
  "FALSE"
)
convertFromCenter(original)
```

---

convertRelationships *Convert pairwise kinship values to relationship categories*

Description

Part of Relations

Usage

```
convertRelationships(kmat, ped, ids = NULL, updateProgress = NULL)
```

Arguments

kmat	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.
ped	datatable that is the Pedigree. It contains pedigree information. The fields id, sire and dam are required.
ids	character vector of IDs or NULL to which the analysis should be restricted. If provided, only relationships between these IDs will be converted to relationships.
updateProgress	function or NULL. If this function is defined, it will be called during each iteration to update a shiny::Progress object.

Details

Ids given as numbers or as a factor, in the pedigree's id, sire and dam or in ids, are taken as their text.

Value

A dataframe with columns id1, id2, kinship, relation. It is a long-form table of pairwise kinships, with relationship categories included for each pair. IDs that are not in kmat are ignored. One ID gives that animal's own Self row, and no ID in kmat (or an empty ids) gives a table with no rows.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
ids <- c("A", "B", "D", "E", "F", "G", "I", "J", "L", "M", "O", "P")
relIds <- convertRelationships(kmat, ped, ids)
rel <- convertRelationships(kmat, ped, updateProgress = function() {})
head(rel)
ped <- nprcgenekeepr::qcPed
bkmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen,
  sparse = FALSE
)
relBIds <- convertRelationships(bkmat, ped, c("4LFS70", "DD1U77"))
relBIds
```

---

convertSexCodes *Convert a sex indicator to a standardized code*

Description

Part of Pedigree Curation

Usage

```
convertSexCodes(sex, ignoreHerm = TRUE)
```

Arguments

sex	character vector or factor of sex codes (see above) for individuals; any other value is treated as unknown.
ignoreHerm	logical flag indicating if hermaphrodites should be treated as unknown sex ("U"), default is TRUE.

Details

Standard sex codes are

- F – replacing "FEMALE" or "2"
- M – replacing "MALE" or "1"
- H – replacing "HERMAPHRODITE" or "4", if ignoreHerm == FALSE
- U – replacing "HERMAPHRODITE" or "4", if ignoreHerm == TRUE
- U – replacing "UNKNOWN" or "3"
- U – replacing a missing, blank or unrecognized value

Case and any spaces around a code are ignored, so " male " and "M " both become M.

Value

A single factor with levels F, M, H and U holding the standardized sex codes after transformation from non-standard codes. Level H is used only when ignoreHerm = FALSE.

Examples

```
library(nprcgenekeepr)
original <- c(
  "m", "male", "1", "MALE", "M", "F", "f", "female",
  "FemAle", "U", "Unknown", "H", "hermaphrodite",
  "U", "Unknown", "3", "4"
)
sexCodes <- convertSexCodes(original)
sexCodes
```

---

convertStatusCodes *Convert status indicators to a standardized code*

Description

Part of Pedigree Curation

Usage

```
convertStatusCodes(status)
```

Arguments

status	character vector or NA. Flag indicating an individual's status as alive, dead, sold, etc.

Details

Case is ignored, but spaces are not trimmed. The recognized codes are ALIVE, A, 1 for alive; DECEASED, DEAD, DIED, D, 2 for deceased; SHIPPED, SHIPED, SOLD, SALE, S, 3 for shipped; and UNKNOWN, U, 4 or NA for unknown.

Value

A factor vector of the standardized status codes with levels: ALIVE, DECEASED, SHIPPED, and UNKNOWN. A value that is not recognized becomes NA, not UNKNOWN.

Examples

```
library(nprcgenekeepr)
original <- c(
  "A", "alive", "Alive", "1", "S", "Sale", "sold", "shipped",
  "D", "d", "dead", "died", "deceased", "2",
  "shiped", "3", "U", "4", "unknown", NA,
  "Unknown", "U", "Unknown", "4"
)
## A value that is not recognized becomes NA
convertStatusCodes("hermaphrodite")
convertStatusCodes(original)
```

---

correctParentSex *Correct the sex of animals listed as a sire or dam*

Description

Part of Pedigree Curation

Usage

```
correctParentSex(id, sire, dam, sex, recordStatus, reportErrors = FALSE)
```

Arguments

id	character vector with unique identifier for an individual
sire	character vector with unique identifier for an individual's father (NA if unknown).
dam	character vector with unique identifier for an individual's mother (NA if unknown).
sex	factor or character vector of sex codes ("M", "F", "H", "U"). Sex specifier for an individual. A character input returns a character vector.
recordStatus	character vector with value of "added" or "original", which indicates whether an animal was added or an original animal. Only "added" is special: an NA, blank or unrecognized value is treated as an original animal, and NULL means no added records are known. It is used only when reportErrors = TRUE.
reportErrors	logical value if TRUE will scan the entire file and make a list of all errors found. The errors will be returned in a list of list where each sublist is a type of error found. When FALSE, an ID listed as both a sire and a dam stops with an error; when TRUE it is returned as sireAndDam.

Details

Only true female-sires (`"F"`) and male-dams (`"M"`) are corrected (to `"M"` and `"F"` respectively). Parents recorded as hermaphrodite (`"H"`) or unknown (`"U"`) sex are left unchanged, consistent with `reportErrors = TRUE` mode, which does not flag them.

When `reportErrors = TRUE`, only records whose `recordStatus` is exactly `"added"` are left out of the report. A missing (`NA`), blank or unrecognized status is treated as an original animal, and a `NULL` `recordStatus` means no added records are known, so every animal is checked.

Value

When `reportErrors = FALSE`, a factor (or character vector) of corrected sex codes with levels `"M"`, `"F"`, `"H"`, and `"U"` for the ids provided. When `reportErrors = TRUE`, a named list of error vectors with elements `sireAndDam`, `femaleSires`, and `maleDams` (each `NULL` when no such errors are found).

Examples

```
library(nprcgenekeepr)
pedOne <- data.frame(
  id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("F", "F", "M", "F", "F", "F", "F", "M"),
  recordStatus = rep("original", 8),
  stringsAsFactors = FALSE
)
pedTwo <- data.frame(
  id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c("d0", "d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("M", "M", "M", "F", "F", "F", "F", "M"),
  recordStatus = rep("original", 8),
  stringsAsFactors = FALSE
)
pedOneCorrected <- pedOne
pedOneCorrected$sex <- correctParentSex(
  pedOne$id, pedOne$sire, pedOne$dam,
  pedOne$sex, pedOne$recordStatus
)
pedOne[pedOne$sex != pedOneCorrected$sex, ]
pedOneCorrected[pedOne$sex != pedOneCorrected$sex, ]

pedTwoCorrected <- pedTwo
pedTwoCorrected$sex <- correctParentSex(
  pedTwo$id, pedTwo$sire, pedTwo$dam,
  pedTwo$sex, pedTwo$recordStatus
)
pedTwo[pedTwo$sex != pedTwoCorrected$sex, ]
pedTwoCorrected[pedTwo$sex != pedTwoCorrected$sex, ]
```

countFirstOrder *Count first-order relatives*

Description

Part of Relations

Usage

```
countFirstOrder(ped, ids = NULL)
```

Arguments

ped	The pedigree information in data.frame format
ids	character vector of IDs or NULL These are the IDs to which the analysis should be restricted. First-order relationships will only be tallied for the listed IDs and will only consider relationships within the subset. If NULL, the analysis will include all IDs in the pedigree.

Details

Tallies the number of first-order relatives for each member of the provided pedigree. If 'ids' is provided, the analysis is restricted to only the specified subset.

Value

A dataframe with column `id`, `parents`, `offspring`, `siblings`, and `total`. A table of first-order relationship counts, broken down to indicate the number of parents, offspring, and siblings that are part of the subset under consideration. Siblings are animals with the same sire and the same dam, so half-siblings are not counted and an animal with an unknown sire or dam has 0 siblings.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ids <- c("B", "D", "E", "F", "G")
countIds <- countFirstOrder(ped, ids)
countIds
count <- countFirstOrder(ped, NULL)
count
```

countKinshipValues *Count kinship-value occurrences across simulated pedigrees*

Description

Count kinship-value occurrences across simulated pedigrees

Usage

```
countKinshipValues(kinshipValues, accummulatedKValueCounts = NULL)
```

Arguments

kinshipValues	data.table of kinship values from simulated pedigrees, as returned by kinshipMatricesToKValues: columns id_1 and id_2 identify a pair of individuals and each remaining column holds the kinship values from one simulated pedigree.
accummulatedKValueCounts	list object with same structure as that returned by this function, from an earlier call. Its ID pairs must be the same, in the same order, as those in kinshipValues; the new counts are added to its counts.

Value

A list of three lists named kIds (kinship IDs), kValues (kinship values), and kCounts (kinship counts).

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_4 <- list(
  id = "J",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
```

```
)
simParent_5 <- list(
  id = "K",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_6 <- list(
  id = "N",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
allSimParents <- list(
  simParent_1, simParent_2, simParent_3,
  simParent_4, simParent_5, simParent_6
)

extractKValue <- function(kValue, id1, id2, simulation) {
  kValue[
    kValue$id_1 == id1 & kValue$id_2 == id2,
    paste0("sim_", simulation)
  ]
}

n <- 10
simKinships <- createSimKinships(ped, allSimParents,
  pop = ped$id, n = n
)
kValues <- kinshipMatricesToKValues(simKinships)
extractKValue(kValues, id1 = "A", id2 = "F", simulation = 1:n)
counts <- countKinshipValues(kValues)

# A second, independent set of simulations is added to the first counts
n <- 10
simKinships <- createSimKinships(ped, allSimParents, pop = ped$id, n = n)
kValues <- kinshipMatricesToKValues(simKinships)
extractKValue(kValues, id1 = "A", id2 = "F", simulation = 1:n)
accummulatedCounts <- countKinshipValues(kValues, counts)
```

`countLoops` *Count the number of loops in a pedigree tree*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
countLoops(loops, ptree)
```

Arguments

loops	a named list of logical values where each named element is named with an id from ptree. The value of the list element is set to TRUE if the id has a loop in the pedigree. Loops occur when an animal's sire and dam have a common ancestor.
ptree	a list of lists forming a pedigree tree as constructed by createPedTree(ped) where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Uses the loops data structure and the list of all ancestors for each individual to calculate the number of loops for each individual.

Value

A list indexed with each ID in the pedigree tree (ptree) containing the number of loops for each individual.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
exampleTree <- createPedTree(examplePedigree)
exampleLoops <- findLoops(exampleTree)
## You can count how many animals are in loops with the following code.
length(exampleLoops[exampleLoops == TRUE])
## You can count how many loops you have with the following code.
nLoops <- countLoops(exampleLoops, exampleTree)
sum(unlist(nLoops[nLoops > 0]))
## You can list the first 10 sets of ids, sires and dams in loops with
## the following line of code:
examplePedigree[exampleLoops == TRUE, c("id", "sire", "dam")][1:10, ]
```

---

createColonySnapshot *Create a colony snapshot row from a genetic value report*

Description

Turns one reportGV result into a one-row data.frame in the 27-column snapshot-history schema (issue #167, schema version 1), ready for appendColonySnapshot. Every metric field is a value the nprcgenekeeprGV object already carries, or a Summary Statistics aggregate (mean/median/calcSkewness/calcKurtosis, NA removed) of its per-animal indivMeanKin/gu/guSE report columns, stored at full precision — no estimator is recomputed or duplicated.

Usage

```
createColonySnapshot(
  ped,
  geneticValue,
  membershipRule,
  guIter,
  guThresh,
  snapshotDate = Sys.Date()
)
```

Arguments

ped	The pedigree data.frame the reportGV analysis was run on, carrying at least an id column (plus the logical population column when membershipRule is "focalPopulation" or "reconstructedLiving").
geneticValue	An object of class nprcgenekeeprGV as returned by reportGV.
membershipRule	Single string naming how the analysis population was assembled; one of "wholePedigree", "focalPopulation" or "reconstructedLiving".
guIter	Single positive whole number: the guIter value the reportGV call used. Required — there is no default.
guThresh	Single positive whole number: the guThresh value the reportGV call used. Required — there is no default.
snapshotDate	The snapshot's date: a Date or an ISO-8601 (YYYY-MM-DD) string. Defaults to Sys.Date().

Details

guIter and guThresh are required because the reportGV return object does not carry them, and they are comparability provenance the snapshot must record truthfully: pass exactly the values the reportGV call used (its defaults are 1000L and 1L).

The claimed membershipRule is verified against ped and the report, and a contradiction stops:

"wholePedigree" the report covers exactly the animals in ped.

"focalPopulation" ped carries the logical population column (see setPopulation) and the report covers exactly ped$id[ped$population]. Pass the same population-designated pedigree the reportGV call analyzed.

"reconstructedLiving" the snapshot was rebuilt for a past date from the pedigree's birth and exit dates (see backfillColonySnapshots). It is verified exactly like "focalPopulation", but is a separate rule so a rebuilt row never mixes with a recorded one: rules are never compared with each other.

Value

A one-row data.frame in the 27-column snapshot-history schema; it passes checkSnapshotHistory unchanged.

Examples

```
ped <- nprcgenekeepr::qcPed
gv <- reportGV(ped, guIter = 10L)
snapshot <- createColonySnapshot(ped, gv, "wholePedigree",
  guIter = 10L, guThresh = 1L
)
history <- appendColonySnapshot(NULL, snapshot)
```

---

createExampleFiles *Create example pedigree and ID-list CSV files*

Description

Creates a folder named `ExamplePedigrees` under the R session temporary directory (as returned by `tempdir()`) if it does not already exist. It then proceeds to write each example data set into a CSV file named based on the name of the data set. The data sets include pedigrees and related lists, such as `focalAnimals`, `qcBreeders`, `pedWithGenotype`, and `qcPed`.

Usage

```
createExampleFiles()
```

Value

A character vector of the full path names of the files written.

Examples

```
library(nprcgenekeepr)
files <- createExampleFiles()
```

---

createPedTree *Create a pedigree tree (PedTree)*

Description

The PedTree is a list containing sire and dam information for an individual.

Usage

```
createPedTree(ped)
```

Arguments

`ped`	datatable that is the `Pedigree`. It contains pedigree information. The fields `id`, `sire` and `dam` are required.

Details

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

This function uses only `id`, `sire`, and `dam` columns.

Value

A list of named lists forming a pedigree tree (PedTree or ptree). Each sublist represents an ID in the pedigree and contains the sire ID and the dam ID as named elements.

Examples

```
library(nprcgenekeepr)
exampleTree <- createPedTree(nprcgenekeepr::examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

`createSimKinships` *Build kinship matrices from simulated pedigrees*

Description

`createSimKinships` uses `makeSimPed` with the `ped` object and the `allSimParents` object to create a set of kinship matrices to be used in forming the *Monte Carlo* estimates for the kinship values.

Usage

```
createSimKinships(
  ped,
  allSimParents,
  pop = NULL,
  n = 10L,
  verbose = FALSE,
  twinRelations = NULL
)
```

Arguments

`ped`	The pedigree information in data.frame format
`allSimParents`	list made up of lists where the internal list has the offspring ID, `id`, a vector of representative sires (`sires`), and a vector of representative dams (`dams`).
`pop`	Character vector with animal IDs to consider as the population of interest. This allows you to provide a pedigree in `ped` that has more animals than you want to use in the simulation by defining `pop` with the subset of interest. The default is NULL.

n	integer value of the number of simulated pedigrees to generate.
verbose	logical vector of length one that indicates whether or not to print a message when a sire or dam is unknown and there are no representative sires or dams to draw from, so it stays NA.
twinRelations	Optional data.frame of declared twin pairs (id1, id2, code) passed straight through to the internal kinship call; only code == "MZ twin" rows affect the computed matrix. NULL (the default) is a complete no-op, matching today's behavior. See kinship's own twinRelations documentation for the full correction mechanism and precondition.

Value

A list of n kinship matrices, each from one simulated pedigree with possible parents drawn for animals with unknown parents.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s2_1", "s2_2", "s2_3"),
  dams = c("d2_1", "d2_2", "d2_3", "d2_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("s3_1", "s3_2", "s3_3"),
  dams = c("d3_1", "d3_2", "d3_3", "d3_4")
)
allSimParents <- list(simParent_1, simParent_2, simParent_3)
pop <- LETTERS[1:7]
simKinships <- createSimKinships(ped, allSimParents, pop, n = 10)
```

---

create_wkbk *Create an Excel workbook with worksheets*

Description

Create an Excel workbook with worksheets

Usage

```
create_wkbk(file, df_list, sheetnames, replace = FALSE)
```

Arguments

file	filename of workbook to be created
df_list	list of data frames to be added as worksheets to workbook
sheetnames	character vector of worksheet names
replace	Specifies if the file should be replaced if it already exist (default is FALSE).

Value

TRUE if the Excel file was successfully created. FALSE, with a warning, if the file already exists and replace is FALSE. Other problems are signaled as errors rather than returned as FALSE; examples are a number of sheetnames that differs from the length of df_list, a folder for file that does not exist and an invalid worksheet name.

Examples

```
library(nprcgenekeepr)

make_df_list <- function(size) {
  df_list <- list(size)
  if (size <= 0) {
    return(df_list)
  }
  for (i in seq_len(size)) {
    n <- sample(2:10, 2, replace = TRUE)
    df <- data.frame(matrix(data = rnorm(n[1] * n[2]), ncol = n[1]))
    df_list[[i]] <- df
  }
  names(df_list) <- paste0("A", seq_len(size))
  df_list
}
df_list <- make_df_list(3)
sheetnames <- names(df_list)
wkbkFile <- file.path(tempdir(), "example_excel_wkbk.xlsx")
create_wkbk(
  file = wkbkFile,
  df_list = df_list,
  sheetnames = sheetnames,
  replace = FALSE
)
if (file.exists(wkbkFile)) {
  file.remove(wkbkFile)
}
```

---

cumulateSimKinships *Compute kinship summary statistics across simulations*

Description

cumulateSimKinships returns a named list of length 4 where the first element is the mean of the simulated kinships, the second element is the standard deviation of the simulated kinships, the third element is the minimum value of the kinships, and the fourth element is the maximum value of the kinships.

Usage

```
cumulateSimKinships(
  ped,
  allSimParents,
  pop = NULL,
  n = 10L,
  twinRelations = NULL
)
```

Arguments

ped	The pedigree information in data.frame format
allSimParents	list made up of lists where the internal list has the offspring ID id, a vector of representative sires (sires), and a vector of representative dams(dams).
pop	Character vector with animal IDs to consider as the population of interest. The default is NULL.
n	integer value of the number of simulated pedigrees to generate. Must be at least 1 (n < 1 is an error); the standard deviation requires n >= 2.
twinRelations	Optional data.frame of declared twin pairs (id1, id2, code) passed straight through to the internal kinship call; only code == "MZ twin" rows affect the computed matrix. NULL (the default) is a complete no-op, matching today's behavior. See kinship's own twinRelations documentation for the full correction mechanism and precondition.

Value

List object containing the meanKinship, sdKinship, minKinship, and maxKinship. sdKinship is the sample standard deviation across the n simulations; it is undefined for a single simulation, so when n < 2 it is returned as NA (with a warning), as base R sd() does for a length-one vector.

Examples

```
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s2_1", "s2_2", "s2_3"),
```

```
  dams = c("d2_1", "d2_2", "d2_3", "d2_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("s3_1", "s3_2", "s3_3"),
  dams = c("d3_1", "d3_2", "d3_3", "d3_4")
)
allSimParents <- list(simParent_1, simParent_2, simParent_3)
pop <- LETTERS[1:7]
cumulativeKinships <- cumulateSimKinships(ped, allSimParents, pop, n = 10)
```

`dataframe2string` *Convert a data frame to a character vector*

Description

Adapted from print.data.frame

Usage

```
dataframe2string(object, ..., digits = NULL, addRowNames = TRUE)
```

Arguments

`object`	dataframe Rows are joined with newline characters, so the result is one string.
`...`	currently unused; accepted for compatibility with print methods.
`digits`	the minimum number of significant digits to be used: see print.default.
`addRowNames`	single logical value. If `TRUE` (the default), the row names of the data frame are printed. Any other value, including a character vector, is treated as `FALSE`.

Value

A character vector of length one holding the text representation of the data.frame provided to the function, with rows separated by newline characters.

Examples

```
library(nprcgenekeepr)
dataframe2string(nprcgenekeepr::pedOne)
```

`drawPedigreeDiagram` *Draw the Pedigree Browser's Diagram from a mating layout*

Description

Returns the interactive diagram the Pedigree Browser's Diagram tab shows, as a `visNetwork` widget, for a layout made by `makePedigreeMatingLayout`. The Shiny application draws its Diagram with this function, so a script that calls it gets the same picture: the sex symbols at the layout's fixed positions, the legend (the five sex symbols, one row per affected condition or a single "Affected" row, "Deceased", and the dashed repeat-appearance and twin lines), each multi-condition symbol split into its colored sections, the slash through each deceased animal's symbol, a canvas that grows with the number of conditions, the **Export Diagram (PNG)** button, a "Select by id" box that lists the animals, and the highlight on hover. The sections and the slash are painted by a drawing script inside the widget, so they appear wherever the widget is shown: in the RStudio viewer, in an R Markdown or Quarto document, or in a web page saved with `htmlwidgets::saveWidget()`.

Usage

```
drawPedigreeDiagram(
  layout,
  edgeStyle = c("rectilinear", "direct"),
  onClick = NULL
)
```

Arguments

`layout`	the list returned by `makePedigreeMatingLayout`.
`edgeStyle`	the `edgeStyle` the layout was made with, `"rectilinear"` (the default) or `"direct"`. It sets how far the hover highlight reaches: 6 steps under `"rectilinear"`, whose invisible routing points count as steps, and 1 under `"direct"`.
`onClick`	`NULL` (the default), or a JavaScript function, as a character string, that runs when the diagram is clicked. The Shiny application passes one that sends the clicked animal to the server.

Details

Clicking an animal re-centers the Shiny application's Diagram on it; that needs a live Shiny session, so a script leaves `onClick` out.

Value

A `visNetwork` widget, 500 pixels tall, 45 more for each affected condition past the first (at most four are drawn).

See Also

`makePedigreeMatingLayout`

Examples

```
library(nprcgenekeepr)
layout <- makePedigreeMatingLayout(nprcgenekeepr::smallPed)
diagram <- drawPedigreeDiagram(layout)
## Print `diagram` in an interactive session to see it, or save it as a web
## page with htmlwidgets::saveWidget(diagram, "pedigree.html").
```

---

exampleNprcgenekeeprConfig *Example nprcgenekeepr configuration file (loadable)*

Description

A loadable version of the example configuration file `example_nprcgenekeepr_config`. It contains a working version of a **nprcgenekeepr** configuration file created at the SNPRC. Users of LabKey's EHR can adapt it to their systems and put it in their home directory. Instructions are embedded as comments within the file.

Usage

```
data(exampleNprcgenekeeprConfig)
```

Format

An object of class `character` of length 89.

Examples

```
library(nprcgenekeepr)
data("exampleNprcgenekeeprConfig")
head(exampleNprcgenekeeprConfig)
```

---

examplePedigree *Example pedigree object (from ExamplePedigree.csv)*

Description

A pedigree object created by `qcStudbook`. Represents pedigree from *ExamplePedigree.csv*.

id – character column of animal IDs

sire – the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam – the female parent of the animal indicated by the `id` column. Unknown dams are indicated with `NA`

sex – factor with levels: "F", "M", "H", "U". Sex specifier for an individual.

gen – generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth – Date vector of birth dates

exit – Date vector of exit dates

age – numerical vector of age in years

ancestry – factor with levels: INDIAN, CHINESE, HYBRID, JAPANESE, OTHER, UNKNOWN indicating the geographic population of origin.

origin – character vector or `NA` (optional) that indicates the name of the facility that the individual was imported from if other than local.

status – character vector or NA. Flag indicating an individual's status as alive, dead, sold, etc. Transformed to factor {levels: ALIVE, DECEASED, SHIPPED, UNKNOWN}. Vector of standardized status codes with the possible values ALIVE, DECEASED, SHIPPED, or UNKNOWN

recordStatus – character vector with value of `"added"` or `"original"`.

fromCenter – logical vector indicating whether the animal was born at the local center (colony-origin), derived from `origin` and `recordStatus`: `TRUE` when `origin` is blank and `recordStatus` is `"original"`; `FALSE` for animals imported from elsewhere (non-blank `origin`) or for synthetic placeholder second-parent rows (`recordStatus == "added"`) whose true origin is not confirmed. Required by `getPotentialParents` to identify in-colony candidate parents.

Usage

```
data(examplePedigree)
```

Format

An object of class `data.frame` with 3694 rows and 13 columns.

Examples

```
library(nprcgenekeepr)
data("examplePedigree")
exampleTree <- createPedTree(examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

`fillGroupMembersWithSexRatio` *Form breeding groups to match a target sex ratio*

Description

The sex ratio is the number of non-males (females and animals of other or unknown sex) per male. A male is placed only if he is at least `minAge` years old; a male younger than that, or with no birth date, is left out of the groups.

Usage

```
fillGroupMembersWithSexRatio(
  candidates,
  groupMembers,
  grpNum,
  kin,
  ped,
  minAge,
  numGp,
  sexRatio
)
```

Arguments

candidates	character vector of IDs of the animals available for use in the group. Ids given as numbers or as a factor, here and in the pedigree's id, sire and dam, are taken as their text.
groupMembers	list initialized and ready to receive groups with the desired sex ratios that are created within this function
grpNum	is a list numGp long with each member a single integer group number, 1 through numGp, as made by makeGroupNum.
kin	list of animals and those animals who are related above a threshold value.
ped	dataframe that is the Pedigree. It contains pedigree information including the IDs listed in ids.
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal younger than this age are ignored; an animal exactly this old is not ignored. There is no default.
numGp	integer value indicating the number of groups that should be formed from the list of IDs. There is no default.
sexRatio	numeric value indicating the number of non-males per male. Values from 0.5 to 20 in steps of 0.5 are typical, but the range is not enforced.

Value

The groupMembers list, with one character vector of animal IDs per group (numGp groups), each filled so that its sex ratio is as close as possible to the ratio specified by sexRatio.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
examplePedigree <- examplePedigree[1:300, ] # Comment out for full example
ped <- qcStudbook(examplePedigree,
  minSireAge = 2L, minDamAge = 2L, reportChanges = FALSE,
  reportErrors = FALSE
)
```

```
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
currentGroups <- list(1)
currentGroups[[1]] <- examplePedigree$id[1L:3L]
candidates <- examplePedigree$id[examplePedigree$status == "ALIVE"]
threshold <- 0.015625
kin <- getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups,
  ignore = list(c("F", "F")), minAge = 1L
)
# Filtering out candidates related to current group members
conflicts <- unique(c(
  unlist(kin[unlist(currentGroups)]),
  unlist(currentGroups)
))
candidates <- setdiff(candidates, conflicts)

kin <- addAnimalsWithNoRelative(kin, candidates)

ignore <- NULL
minAge <- 1.0
numGp <- 1L
harem <- FALSE
sexRatio <- 0.0
withKin <- FALSE
groupMembers <- nprcgenekeepr::makeGroupMembers(numGp,
  currentGroups,
  candidates,
  ped,
  harem = harem,
  minAge = minAge
)
groupMembersStart <- groupMembers
grpNum <- nprcgenekeepr::makeGroupNum(numGp)

groupMembers <- fillGroupMembersWithSexRatio(
  candidates, groupMembers, grpNum, kin, ped, minAge, numGp,
  sexRatio = 1.0
)
```

`filterKinMatrix` *Filter a kinship matrix to selected IDs*

Description

Filter a kinship matrix to selected IDs

Usage

```
filterKinMatrix(ids, kmat)
```

Arguments

ids	character vector containing the IDs of interest. The kinship matrix should be reduced to only include these rows and columns.
kmat	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.

Value

A numeric matrix that is the reduced kinship matrix with named rows and columns (row and col names are 'ids'). It is a matrix however many IDs match: 1 x 1 for one ID, and with no rows or columns when none match.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen,
  sparse = FALSE
)
ids <- ped$id[c(189, 192, 194, 195)]
ncol(kmat)
nrow(kmat)
kmatFiltered <- filterKinMatrix(ids, kmat)
ncol(kmatFiltered)
nrow(kmatFiltered)
```

---

filterPairs *Filter kinship pairs by the animals' sexes*

Description

Part of Group Formation

Usage

```
filterPairs(
  kin,
  ped,
  ignore = list(c(sexCodes[["female"]], sexCodes[["female"]]))
)
```

Arguments

kin	a dataframe with columns id1, id2, and kinship. This is the kinship data reformatted from a matrix, to a long-format table.

ped	Dataframe of pedigree information that must contain an id column and a sex column. The id values should include the animals referenced in kin. Pairs involving an animal with a missing sex, or one absent from ped, can be silently removed from the result.
ignore	a list containing zero or more character vectors of length 2 indicating which sex pairs should be ignored with regard to kinship. Defaults to list(c("F", "F")).

Value

A dataframe representing a filtered long-format kinship table.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
kin <- kinMatrix2LongForm(kmat, removeDups = FALSE)
threshold <- 0.1
kin <- filterThreshold(kin, threshold = threshold)
ped$sex <- c("M", "F", "M", "M", "F", "F", "M")
kinNull <- filterPairs(kin, ped, ignore = NULL)
kinMM <- filterPairs(kin, ped, ignore = list(c("M", "M")))
ped
kin[kin$id1 == "C", ]
kinMM[kinMM$id1 == "C", ]
```

---

filterReport *Filter a genetic value report to selected animals*

Description

Filter a genetic value report to selected animals

Usage

```
filterReport(ids, rpt)
```

Arguments

ids	character vector of animal IDs
rpt	a dataframe with the required colname id, such as the data.frame of results from a genetic value analysis. Only id is used; all other columns are returned unchanged.

Value

A copy of report specific to the specified animals.

Examples

```
library(nprcgenekeepr)
rpt <- nprcgenekeepr::pedWithGenotypeReport$report
rpt1 <- filterReport(c("GHH9LB", "BD41WW"), rpt)
```

`filterThreshold` *Filter out kinship pairs below a threshold*

Description

Part of Group Formation Filters kinship values less than the specified threshold from a long-format table of kinship values.

Usage

```
filterThreshold(kin, threshold = 0.015625)
```

Arguments

`kin`	a dataframe with columns `id1`, `id2`, and `kinship`. This is the kinship data reformatted from a matrix, to a long-format table.
`threshold`	numeric value representing the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored.

Value

The filtered long-format kinship table (a data.frame) with all kinship relationships below the threshold value removed.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
kin <- kinMatrix2LongForm(kmat, removeDups = FALSE)
kinFiltered_0.3 <- filterThreshold(kin, threshold = 0.3)
kinFiltered_0.1 <- filterThreshold(kin, threshold = 0.1)
```

finalRpt *Genetic-value report list prior to ranking*

Description

A list object created from the list object *rpt* prepared by `reportGV`. It is created inside `orderReport`. This version is at the state just prior to calling `rankSubjects` inside `orderReport`.

Usage

```
data(finalRpt)
```

Format

An object of class `list` of length 3.

Details

It is a list of three data frames: `highGu`, `lowMk` and `lowVal`. Each has the 13 columns `id`, `sex`, `age`, `birth`, `exit`, `population`, `first_name`, `second_name`, `indivMeanKin`, `zScores`, `gu`, `totalOffspring` and `livingOffspring`.

Examples

```
library(nprcgenekeepr)
data("finalRpt")
finalRpt <- rankSubjects(finalRpt)
```

findGeneration *Determine the generation number for each ID*

Description

This loops through the entire pedigree one generation at a time. It finds the zeroth generation during first loop. The first time through this loop no sire or dam is in parents. This means that the animals without a sire and without a dam are assigned to generation 0 and become the first parental generation. The second time through this loop finds all of the animals that do not have a sire or do not have a dam and at least one parent is in the vector of parents defined the first time through. The ids that were not assigned as parents in the previous loop are given the incremented generation number.

Usage

```
findGeneration(id, sire, dam)
```

Arguments

id	character vector with unique identifier for an individual
sire	character vector with unique identifier for an individual's father (NA if unknown).
dam	character vector with unique identifier for an individual's mother (NA if unknown).

Details

Subsequent trips in the loop repeat what was done the second time through until no further animals can be added to the nextGen vector.

This does not work if the pedigree does not have all parent IDs as ego IDs.

Value

An integer vector indication the generation numbers for each id, starting at 0 for individuals lacking IDs for both parents. Any id that cannot be placed — e.g. when the pedigree contains a cycle or references a parent ID that is not itself present as an ego ID — is returned as NA and triggers a warning naming the affected ids.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped[, c("id", "sire", "dam")]
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
ped
```

---

findLoops *Find loops in a pedigree tree*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
findLoops(ptree)
```

Arguments

ptree	a list of lists forming a pedigree tree as constructed by createPedTree(ped) where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Value

A named list of logical values where each named element is named with an `id` from `ptree`. The value of the list element is set to `TRUE` if the `id` has a loop in the pedigree. Loops occur when an animal's sire and dam have a common ancestor.

Examples

```
data("examplePedigree")
exampleTree <- createPedTree(examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

`findOffspring` *Count total offspring for each animal*

Description

Part of Genetic Value Analysis

Usage

```
findOffspring(probands, ped)
```

Arguments

`probands`	character vector of egos for which offspring should be counted and returned.
`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam). This requires complete pedigree information. Ids given as numbers or as a factor in `sire` and `dam` are taken as their text.

Value

A named integer vector containing the offspring counts for each animal in `probands`. The names are set to the IDs from `probands`, and an ID that is not a parent in `ped` gets 0.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2,
  minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
```

```
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
totalOffspring <- findOffspring(probands, ped)
```

`findPedigreeNumber` *Determine the pedigree number for each ID*

Description

One of Pedigree Curation functions

Usage

```
findPedigreeNumber(id, sire, dam)
```

Arguments

`id`	character vector with unique identifier for an individual
`sire`	character vector with unique identifier for an individual's father (`NA` if unknown).
`dam`	character vector with unique identifier for an individual's mother (`NA` if unknown).

Value

Integer vector indicating the pedigree (family group) number for each id. Ids that are connected through parent-offspring links share the same number; numbering starts at 1. Numbering starts from ids with both parents unknown (`NA`), so an id that cannot be reached from such an id, for example one whose only parent is absent from `id`, or one in a cycle, is returned as `NA`.

Examples

```
library(nprcgenekeepr)
library(stringi)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- NULL
ped$population <- NULL
ped2 <- ped
ped2$id <- stri_c(ped$id, "2")
ped2$sire <- stri_c(ped$sire, "2")
ped2$dam <- stri_c(ped$dam, "2")
ped3 <- ped
ped3$id <- stri_c(ped$id, "3")
ped3$sire <- stri_c(ped$sire, "3")
ped3$dam <- stri_c(ped$dam, "3")
ped <- rbind(ped, ped2)
ped <- rbind(ped, ped3)
ped$pedigree <- findPedigreeNumber(ped$id, ped$sire, ped$dam)
ped$pedigree
```

fixColumnNames *Standardize pedigree column names*

Description

Standardizing converts the names to lower case, removes spaces, periods and underscores, and then renames by substring: `egoid` and `ego` become `id`, `sireid` becomes `sire`, `damid` becomes `dam`, `birthdate` becomes `birth` and `deathdate` becomes `death`. `recordstatus`, `fromcenter` and `geographicorigin` are restored to their camel case forms; a header already spelled `recordStatus`, `fromCenter` or `geographicOrigin` is not recorded as a change. The headers `first_name` and `second_name` keep their underscore. The matching is unanchored, so any name that contains one of these strings is changed (for example, `category` becomes `catidry`). A header that is `NA` is read as a blank one, so a nameless column is reported the way a column with a blank header is.

Usage

```
fixColumnNames(orgCols, errorLst)
```

Arguments

`orgCols`	character vector with ordered list of column names found in a pedigree file. A name that is `NA` counts as blank.
`errorLst`	list object with places to store the various column name changes, as returned by `getEmptyErrorLst`.

Value

A list object with `newColNames` and `errorLst` with a record of all changes made.

Examples

```
library(nprcgenekeepr)
fixColumnNames(c("Sire_ID", "EGO", "DAM", "Id", "birth_date"),
  errorLst = getEmptyErrorLst()
)
```

focalAnimals *Focal animal IDs from examplePedigree*

Description

A dataframe with one column (*id*) containing the animal Ids from the **examplePedigree** pedigree. They can be used to illustrate the identification of a population of interest as is shown in the example below. The `id` column is a factor, not a character vector like the `id` column of `examplePedigree`; use `as.character(focalAnimals$id)` when the two need the same type.

Usage

```
data(focalAnimals)
```

Format

An object of class `data.frame` with 327 rows and 1 columns.

Examples

```
library(nprcgenekeepr)
data("focalAnimals")
data("examplePedigree")
any(names(examplePedigree) == "population")
nrow(examplePedigree)
examplePedigree <- setPopulation(
  ped = examplePedigree,
  ids = focalAnimals$id
)
any(names(examplePedigree) == "population")
nrow(examplePedigree)
nrow(examplePedigree[examplePedigree$population, ])
```

---

geneDrop *Simulate gene dropping through a pedigree*

Description

Part of Genetic Value Analysis

Usage

```
geneDrop(
  ids,
  sires,
  dams,
  gen,
  genotype = NULL,
  n = 1000L,
  updateProgress = NULL
)
```

Arguments

`ids`	A character vector of unique IDs for a set of animals.
`sires`	A character vector with IDS of the sires for the set of animals. `NA` is used for missing sires.
`dams`	A character vector with IDS of the dams for the set of animals. `NA` is used for missing dams.

gen	An integer vector indicating the generation number for each animal.
genotype	A dataframe containing known genotypes. It has three columns: id, first, and second. The second and third columns contain the integers indicating the observed genotypes. The codes must not collide with the integer codes the function assigns to unknown founder alleles (1, 2, 3, ...).
n	integer indicating the number of iterations to simulate. Default is 1000.
updateProgress	function or NULL. If this function is defined, it will be called once at the start (with reset = TRUE) and then once for each animal processed, to update a shiny::Progress object.

Details

The gene dropping method from *Pedigree analysis by computer simulation* by Jean W MacCluer, John L Vandeberg, and Oliver A Ryder (1986) doi:10.1002/zoo.1430050209 is used in the genetic value calculations.

Currently there is no means of handling knowing only one haplotype. Another column could be added to handle situations where only one allele is observed and it is not known to be homozygous or heterozygous. The new fourth column could have a frequency for homozygosity that could be used in the gene dropping algorithm.

The genotypes are using indirection (integer instead of character) to indicate the genes because the manipulation of character strings was found to take 20-35 times longer to perform.

Adding additional columns to genotype does not significantly affect the time require. Thus, it is convenient to add the corresponding haplotype names to the dataframe using first_name and second_name.

Animal IDs (ids) must not contain a period ("."). A period is disallowed because it causes problems across software environments (R column-name and formula parsing, file-name extensions, programming-language namespaces, and regular expressions); geneDrop additionally relies on the period internally to recover the id and parent of each allele row, so a period-bearing id would silently corrupt the result. IDs containing a period are therefore rejected with an error. The same rule is enforced at data input by qcStudbook and honored by all automatically generated IDs.

Animal IDs (ids) must also be unique. geneDrop indexes each animal's parents and accumulates its simulated alleles by id, so duplicate ids are rejected with an error. This invariant is established upstream by qcStudbook (via removeDuplicates) and by kinship, both of which require unique ids.

Value

A data.frame V1 ... Vn, id, parent A data.frame providing the maternal and paternal alleles for an animal for each iteration. The first n columns indicate the allele for each iteration. These are followed by two columns: id, the animal's ID, and parent, whether the allele came from the sire ("sire") or dam ("dam"). Each animal has two rows, and the animals are ordered by generation, oldest first.

Examples

```
## We usually define `n` to be >= 1000
library(nprcgenekeepr)
```

```
ped <- nprcgenekeepr::lacy1989Ped
allelesNew <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = NULL, n = 50, updateProgress = NULL
)
genotype <- data.frame(
  id = ped$id,
  first_allele = c(
    NA, NA, "A001_B001", "A001_B002",
    NA, "A001_B002", "A001_B001"
  ),
  second_allele = c(
    NA, NA, "A010_B001", "A001_B001",
    NA, NA, NA
  ),
  stringsAsFactors = FALSE
)
pedWithGenotype <- addGenotype(ped, genotype)
pedGenotype <- getGVGenotype(pedWithGenotype)
allelesNewGen <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = pedGenotype,
  n = 5, updateProgress = NULL
)
```

`getAncestors` *Recursively collect an individual's ancestors*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
getAncestors(id, ptree)
```

Arguments

`id`	character vector of length 1 having the ID of interest
`ptree`	a list of lists forming a pedigree tree as constructed by `createPedTree(ped)` where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

An ancestor reached through both parents is returned once for each route. The function stops with an error that names the ids involved when `ptree` contains a cycle (an animal that is its own ancestor), which is a data error.

A sire or dam that has no entry of its own in `ptree` is treated as an animal with unknown parents, as `qcStudbook()` treats it (`addParents()` adds the entry): it is returned as an ancestor and has no ancestors of its own. The function stops with an error that names the id when `id` itself is not in `ptree`.

Value

A character vector of ancestors for an individual ID.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped <- qcStudbook(ped, minSireAge = 0, minDamAge = 0)
pedTree <- createPedTree(ped)
pedLoops <- findLoops(pedTree)
ids <- names(pedTree)
allAncestors <- list()

for (i in seq_along(ids)) {
  id <- ids[[i]]
  anc <- getAncestors(id, pedTree)
  allAncestors[[id]] <- anc
}
head(allAncestors)
countOfAncestors <- unlist(lapply(allAncestors, length))
idsWithMostAncestors <-
  names(allAncestors)[countOfAncestors == max(countOfAncestors)]
allAncestors[idsWithMostAncestors]
```

---

getAnimalsWithHighKinship

List each animal's high-kinship relatives

Description

List each animal's high-kinship relatives

Usage

```
getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups, ignore, minAge)
```

Arguments

`kmat`	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.
`ped`	The pedigree information in data.frame format
`threshold`	numeric value representing the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored.

currentGroups	list of character vectors of IDs of animals currently assigned to the group. Required (no default); use `character(0L)` when no groups exist.
ignore	list of character vectors representing the sex combinations to be ignored. If provided, the vectors in the list specify if pairwise kinship should be ignored between certain sexes. Required (no default); `filterPairs()` itself ignores female-female pairs when called directly.
minAge	integer value indicating the minimum age to consider in group formation. Required (no default). Pairwise kinships involving an animal younger than this age are ignored; animals of exactly this age or with a missing age are retained.

Value

A one-dimensional array of mode list (from `tapply()`), not a plain list. Its names are animal IDs, and each element is a character vector of animals sharing a kinship value greater than or equal to the `threshold` value. `names()` and `[[` work as for a list.

Examples

```
qcPed <- nprcgenekeepr::qcPed
ped <- qcStudbook(qcPed,
  minSireAge = 2L, minDamAge = 2L, reportChanges = FALSE,
  reportErrors = FALSE
)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
currentGroups <- list(1L)
currentGroups[[1L]] <- examplePedigree$id[1L:3L]
candidates <- examplePedigree$id[examplePedigree$status == "ALIVE"]
threshold <- 0.015625
kin <- getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups,
  ignore = list(c("F", "F")), minAge = 1.0
)
length(kin) # should be 259
kin[["0DAV0I"]] # should have 34 IDs
```

---

getAutoIdFormat *Get the auto-generated unknown-ID format*

Description

Returns the `sprintf` template used to mint placeholder IDs for unknown parents (see `addUIds`) and to detect them. The format is the single source of truth shared by ID *generation* and ID *detection*.

Usage

```
getAutoIdFormat()
```

Details

The value is read from `getOption("nprcgenekeepr.autoIdFormat")`, defaulting to `"U%04d"` (a leading `"U"` plus a zero-padded integer) for backward compatibility. Set it with `setAutoIdFormat`.

Value

A single character string: the auto-ID `sprintf` format.

See Also

`setAutoIdFormat`, `addUIds`

Examples

```
getAutoIdFormat()
```

---

getBoxWhiskerDescription

Get Box and Whisker Plot Description

Description

Returns a description of how box and whisker plots work, suitable for use in popovers and tooltips in the Shiny application.

Usage

```
getBoxWhiskerDescription()
```

Value

Character string containing the box and whisker plot description explaining whiskers, IQR (interquartile range), and outliers.

See Also

`modSummaryStatsServer` which uses this for popovers

Examples

```
desc <- getBoxWhiskerDescription()
cat(desc)
```

getChangedColsTab *Build the changed-columns tab panel*

Description

Build the changed-columns tab panel

Usage

```
getChangedColsTab(errorLst, pedigreeFileName)
```

Arguments

errorLst	list of errors and changes made by qcStudbook
pedigreeFileName	name of file provided by user on Input tab

Value

A Shiny tabPanel titled "Changed Columns" that holds the HTML-formatted list of column changes made by qcStudbook.

getConfigFileName *Get the configuration file name for the system*

Description

Get the configuration file name for the system

Usage

```
getConfigFileName(sysInfo)
```

Arguments

sysInfo	object returned by Sys.info()

Value

A named character vector of length two: homeDir is the user's home directory and configFile is the expected configuration file path. Only sysInfo[["sysname"]] is used, to choose the file name (_nprcgenekeepr_config on Windows, otherwise .nprcgenekeepr_config).

Examples

```
library(nprcgenekeepr)
sysInfo <- Sys.info()
config <- getConfigFileName(sysInfo)
```

getCurrentAge *Calculate current age in years from a birth date*

Description

Assumes current date for calculating age.

Usage

```
getCurrentAge(birth)
```

Arguments

birth	birth date(s)

Value

Age in years using the provided birthdate.

Examples

```
library(nprcgenekeepr)
age <- getCurrentAge(birth = as.Date("06/02/2000", format = "%m/%d/%Y"))
```

getDatedFilename *Prepend the date and time to a file name*

Description

Prepend the date and time to a file name

Usage

```
getDatedFilename(filename)
```

Arguments

filename	character vector with name to use in file name

Value

A character string with `filename` prepended with the current date and time. The prefix is `as.character()` of the current time with spaces and colons replaced by underscores; it can include fractional seconds, for example `2026-10-01_15_39_25.959408_testName`.

Examples

```
library(nprcgenekeepr)
getDatedFilename("testName")
```

`getDateErrorsAndConvertDatesInPed` *Find date errors and convert dates in a pedigree*

Description

Finds date errors in columns defined in `convertDate` as dates and converts date strings to `Date` objects.

Usage

```
getDateErrorsAndConvertDatesInPed(sb, errorLst)
```

Arguments

`sb`	A dataframe containing a table of pedigree and demographic information.
`errorLst`	object with placeholders for error types found in a pedigree file by `qcStudbook` through the functions it calls.

Details

Every date column is checked on its own and each invalid date is listed by row; the invalid dates become `NA` in the converted pedigree, and exit dates are calculated and added, unless every original record has an invalid date, in which case the pedigree is returned unchanged.

Value

A list with the pedigree, `sb`, and the `errorLst` with invalid date rows (`errorLst$invalidDateRows`), a character vector of row numbers (`character(0)` when all dates are valid). When the dates are converted, `sb` also gains an `exit` column.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedInvalidDates
ped
errorLst <- getEmptyErrorLst()
colNamesAndErrors <- fixColumnNames(names(ped), errorLst)
names(ped) <- colNamesAndErrors$newColNames
pedAndErrors <- getDateErrorsAndConvertDatesInPed(ped, errorLst)
pedAndErrors$sb
pedAndErrors$errorLst
```

getDemographics *Get demographic data*

Description

This is a thin wrapper around `labkey.selectRows()`. It uses the site configuration from `getSiteInfo` (the configuration file, or the defaults plus a warning when there is no file) and sets the LabKey credentials with `setLabKeyDefaults`. All columns are returned, including hidden ones, when `colSelect` is `NULL`.

Usage

```
getDemographics(colSelect = NULL)
```

Arguments

`colSelect`	(optional) a vector of comma separated strings specifying which columns of a dataset or view to import

Value

A data.frame containing LabKey demographic data with the columns specified in the single parameter provided.

Examples

```
## Not run:
## Needs a connection to a LabKey server
library(nprcgenekeepr)
siteInfo <- getSiteInfo()
colSet <- siteInfo$lkPedColumns
source <- " generated by getDemographics: "
pedSourceDf <- tryCatch(getDemographics(colSelect = colSet),
  warning = function(wCond) {
    cat(paste0("Warning", source, wCond),
      name = "nprcgenekeepr"
    )
    return(NULL)
  },
  error = function(eCond) {
    cat(paste0("Error", source, eCond),
      name = "nprcgenekeepr"
    )
    return(NULL)
  }
)

## End(Not run)
```

getDescendantPedigree *Reduce a pedigree to a group and its descendants*

Description

Filters a pedigree down to only the descendants of the provided group, building the pedigree forward in time starting from a group of probands. This is the downward (descendants-only) mirror of `getProbandPedigree`: it takes the transitive closure over offspring and returns the probands together with all of their descendants. It does not include collateral relatives (siblings, cousins, or mates).

Usage

```
getDescendantPedigree(probands, ped)
```

Arguments

`probands`	a character vector with the list of animals whose descendants should be included in the final pedigree.
`ped`	datatable that is the `Pedigree`. It contains pedigree information. The fields `id`, `sire` and `dam` are required.

Value

A reduced pedigree containing the probands and all of their descendants.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
## D's descendants are F and G
getDescendantPedigree(probands = "D", ped = ped)$id
```

getEmptyErrorLst *Create an empty errorLst object*

Description

Create an empty errorLst object

Usage

```
getEmptyErrorLst()
```

Value

An errorLst object (a list of class `nprcgenekeeprErr`) with placeholders for error types found in a pedigree file by `qcStudbook`. Its 11 fields are `failedDatabaseConnection`, `missingColumns`, `invalidDateRows`, `suspiciousParents`, `femaleSires`, `maleDams`, `sireAndDam`, `duplicateIds`, `invalidIdChars`, `invalidPlaceholderRows` and `changedCols` (a list of the column-name changes).

Examples

```
library(nprcgenekeepr)
getEmptyErrorLst()
```

---

getErrorTab *Build the error-list tab panel*

Description

Build the error-list tab panel

Usage

```
getErrorTab(errorLst, pedigreeFileName)
```

Arguments

`errorLst`	list of errors and changes made by `qcStudbook`
`pedigreeFileName`	name of file provided by user on Input tab

Value

A Shiny `tabPanel` titled "Error List" that holds the HTML-formatted list of errors found by `qcStudbook`.

---

getFileDirectRelatives *Get the direct relatives of selected animals from a pedigree file*

Description

File-sourced sibling of `getLkDirectRelatives`: reads a pedigree file through the internal `getPedigreeSource()` `"file"` provider (via `getPedigree`), then delegates the pedigree walk to the source-agnostic `getPedDirectRelatives`. The result is the full connected pedigree component (ancestors, descendants, and collaterals such as siblings and mates) reachable from the focal animals. It is fully offline and deterministic.

Usage

```
getFileDirectRelatives(
  ids,
  fileName = NULL,
  sep = ",",
  unrelatedParents = FALSE
)
```

Arguments

ids	character vector of animal IDs
fileName	path to a pedigree file (CSV or Excel) read via getPedigree; the file must provide at least id, sire, and dam columns.
sep	column separator passed to the file reader for delimited text files (default ","); ignored for Excel files.
unrelatedParents	logical vector when FALSE the unrelated parents of offspring do not get a record as an ego; when TRUE a place holder record where parent (sire, dam) IDs are set to NA.

Details

Unlike the LabKey source, which fails soft (returns NULL) when its fetch fails, the file source errors loudly: a NULL or missing fileName, a file that does not exist, or a file lacking the id, sire, and dam columns each raises an error.

Value

A data.frame with pedigree structure containing all direct relatives – the full connected pedigree component (ancestors, descendants, and collaterals) – for the Ids provided.

See Also

Other direct relatives: getLkDirectAncestors(), getLkDirectRelatives(), getPedDirectRelatives()

Examples

```
library(nprcgenekeepr)
## Build a tiny pedigree file, then pull the relatives of a focal animal.
ped <- data.frame(
  id = c("A", "B", "C"), sire = c(NA, NA, "A"), dam = c(NA, NA, "B"),
  stringsAsFactors = FALSE
)
pedFile <- tempfile(fileext = ".csv")
write.csv(ped, pedFile, row.names = FALSE)
getFileDirectRelatives(ids = "C", fileName = pedFile)
unlink(pedFile)
```

`getFocalAnimalPed` *Get pedigree based on list of focal animals*

Description

Reads the focal animal ids from the first column of a CSV or Excel file and pulls the pedigree of their relatives from LabKey.

Usage

```
getFocalAnimalPed(fileName, sep = ",")
```

Arguments

`fileName`	character vector of temporary file path of a CSV or Excel file whose first column lists the focal animal ids.
`sep`	column separator in CSV file

Value

A data.frame (not a file) with columns `id`, `sex`, `birth`, `death`, `departure`, `dam` and `sire`; the dates are character strings formatted as `%Y-%m-%d`. When the LabKey connection fails, an `nprcgenekeeprErr` list with `failedDatabaseConnection` set is returned instead.

Examples

```
library(nprcgenekeepr)
## A successful call needs a LabKey connection and a file of focal ids.
## This file does not exist, so the example shows the error handler.
source <- " generated by getFocalAnimalPed: "
tryCatch(getFocalAnimalPed(fileName = "breeding file.csv"),
  warning = function(wCond) {
    cat(paste0("Warning", source, wCond),
      name = "nprcgenekeepr"
    )
    return(NULL)
  },
  error = function(eCond) {
    cat(paste0("Error", source, eCond),
      name = "nprcgenekeepr"
    )
    return(NULL)
  }
)
```

`getFocalAnimalPedFromFile` *Get a focal-animal pedigree from a pedigree file*

Description

File-sourced sibling of `getFocalAnimalPed`: reads a list of focal animal Ids from `fileName` (the first column, exactly as `getFocalAnimalPed` does), then builds the focal animals' full connected pedigree component from a SEPARATE pedigree file via `getFileDirectRelatives`. This lets the focal-animal workflow run entirely offline – no LabKey / EHR connection is required.

Usage

```
getFocalAnimalPedFromFile(fileName, pedigreeFileName = NULL, sep = ",")
```

Arguments

`fileName`	character path to a file (CSV, delimited text, or Excel) whose first column is the list of focal animal Ids.
`pedigreeFileName`	character path to the pedigree file (CSV, delimited text, or Excel) read via `getPedigree`; it must provide at least `id`, `sire`, and `dam` columns.
`sep`	column separator passed to the file readers for delimited text files (default `","`); ignored for Excel files.

Details

The underlying file source errors loudly on a bad pedigree file, but this function is the application boundary, so it is fail-soft: it does NOT throw. On failure it returns a classed `nprcgenekeeprFileErr` object whose `message` names the reason – a missing, not-found, or unreadable pedigree file, or one lacking the `id`, `sire`, and `dam` columns. (This mirrors how the app's other file inputs behave – the `message` surfaces as a "File Read Error" – and is distinct from the LabKey path, which returns an `nprcgenekeeprErr`.)

Value

On success, a data.frame with the focal animals' full connected pedigree component (ancestors, descendants, and collaterals), as returned by `getFileDirectRelatives`. On any failure this function does NOT throw: it returns a classed `nprcgenekeeprFileErr` object (a list with a `message` element) naming WHY the read failed – an unreadable focal-id list file; a missing, not-found, unreadable, or wrong-column pedigree file; or no focal IDs present in the pedigree. The application surfaces `message` as the "File Read Error" detail (distinct from the LabKey path, which returns an `nprcgenekeeprErr`).

Examples

```
library(nprcgenekeepr)
## A focal-id file and a pedigree file, then build the pedigree offline.
ped <- data.frame(
  id = c("A", "B", "C"), sire = c(NA, NA, "A"), dam = c(NA, NA, "B"),
  stringsAsFactors = FALSE
)
pedFile <- tempfile(fileext = ".csv")
write.csv(ped, pedFile, row.names = FALSE)
focalFile <- tempfile(fileext = ".csv")
write.csv(data.frame(id = "C"), focalFile, row.names = FALSE)
getFocalAnimalPedFromFile(focalFile, pedFile)
unlink(c(pedFile, focalFile))
```

---

getFounders *Get the founder ids from a pedigree*

Description

Part of Pedigree Curation

Usage

```
getFounders(ped)
```

Arguments

ped	datatable that is the Pedigree. It contains pedigree information. The fields id, sire and dam are required.

Details

A founder is an animal whose sire and dam are both unknown (NA). Animals with exactly one known parent (partial parentage) are **not** founders.

Value

A vector of the id values of the founders, in pedigree order. It has the same type as ped$id and is empty when there are no founders.

See Also

isFounder for the founder logical mask.

Examples

```
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
getFounders(ped)
```

`getGeneticDiversityStats` *Assemble breeding-group genetic diversity heat-map statistics*

Description

Assemble breeding-group genetic diversity heat-map statistics

Usage

```
getGeneticDiversityStats(
  groups,
  ped,
  geneticValues,
  kmat,
  housing = "shelter_pens",
  currentDate = Sys.Date()
)
```

Arguments

`groups`	List of character vectors of animal IDs, one per breeding group (for example the `groups` returned by `modBreedingGroupsServer`). If the list is named, the names become the group row labels; otherwise labels are `"Group 1"`, `"Group 2"`, and so on.
`ped`	Dataframe that is the `Pedigree`. The `id`, `dam`, `sex`, `birth`, and `exit` columns are required; an optional `ancestry` column enables the Origin metric.
`geneticValues`	Dataframe of the genetic value report (for example `reportGV(ped)$report`). The `id` and `value` columns are required; `value` holds the labels produced by `rankSubjects` (`"Low Value"`, `"High Value"`, `"Undetermined"`).
`kmat`	Square kinship matrix whose row and column names are animal IDs and that covers every member of every group (for example the matrix returned by `kinship`).
`housing`	Character housing type passed to `getProductionStatus`, either `"shelter_pens"` or `"corral"`. Length 1 (applied to every group) or one value per group. Defaults to `"shelter_pens"`.
`currentDate`	Date used to derive age and the production birth window. Defaults to `Sys.Date()`.

Details

For each breeding group this builds the heat-map color indices by calling the per-group providers: Value from `getProportionLow`, Origin from `getIndianOriginStatus`, Production from `getProductionStatus`, and Inbreeding from `getKinshipWithMaleStatus`. The result is the group-by-metric data frame consumed by `makeGeneticDiversityHeatmap`: the first column holds the group label and each remaining column holds a color index (1 red, 2 yellow, 3 green).

Age is derived from each member's birth date using `currentDate` rather than read from a possibly-absent `age` column, so the age filters and the production birth window share one reference date. Genetic-value labels of `"Undetermined"` are dropped before the Value proportion is taken. A group with no assessed value, and a group whose Inbreeding metric is undefined (no breeding-age females), are both scored red so that missing data is surfaced rather than shown as healthy green. When the pedigree has no `ancestry` column the Origin metric cannot be computed and its column is omitted.

Production is the number of births in the two calendar years ending the year before `currentDate` that lived at least 30 days, divided by the number of breeding-age females in the group: females at least 3 years old. A female with no birth date has no age. She counts as a breeding-age female only when the pedigree (`ped`) lists at least one animal as her offspring, looked up in the whole pedigree and not only in her group, so a mother whose offspring are in another group still counts. A female with no birth date and no offspring is left out, and so is an animal with a blank sex. A group with no breeding-age females has no Production value: its entry is `NA`, and the heat map draws that cell gray.

The Inbreeding column counts breeding-age females by the same rule: a female with no birth date counts when the pedigree lists an offspring for her, and is left out otherwise. A male with no birth date is never counted as a potential mate.

Value

A data frame with one row per group: the first column `group` holds the group label and each remaining column (`Value`, `Origin` when available, `Production`, `Inbreeding`) holds an integer color index in `c(1, 2, 3)`. `Production` is `NA` for a group with no breeding-age females.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
## The genetic value report of qcPed, stored with the package
geneticValues <- nprcgenekeepr::qcPedGvReport$report
## Three groups of living animals that have a genetic value
living <- geneticValues$id[geneticValues$id %in% ped$id[is.na(ped$exit)]]
groups <- list(
  "Group 1" = living[1L:15L],
  "Group 2" = living[16L:30L],
  "Group 3" = living[31L:45L]
)
getGeneticDiversityStats(groups, ped, geneticValues, kmat)
```

getGenotypes *Get genotypes from file*

Description

Get genotypes from file

Usage

```
getGenotypes(fileName, sep = ",")
```

Arguments

fileName	character vector of length one: the path of a delimited text file or an Excel (xls/xlsx) file.
sep	column separator in a delimited text file; ignored for an Excel file.

Value

The file's contents as an unchecked dataframe (column names are not changed). In a delimited text file an empty string and "NA" are read as NA. Pass the result to checkGenotypeFile to make it compatible with the other functions in this package.

Examples

```
library(nprcgenekeepr)
pedCsv <- getGenotypes(fileName = system.file("testdata", "qcPed.csv",
  package = "nprcgenekeepr"
))
```

getGVGenotype *Extract genotype data for a genetic value report*

Description

Extracts genotype data if available otherwise NULL is returned.

Usage

```
getGVGenotype(ped)
```

Arguments

ped	The pedigree information in data.frame format

Value

A data.frame with the columns id, first, and second extracted from a pedigree object (a data.frame) containing genotypic data. If the pedigree object does not contain genotypic data (columns id, first and second, with numeric first and second), NULL is returned.

Examples

```
## We usually define `n` to be >= 1000. The first part of this example is
## the same as the one for `geneDrop()`; `getGVGenotype()` is used on the
## last lines.
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
allelesNew <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = NULL, n = 50, updateProgress = NULL
)
genotype <- data.frame(
  id = ped$id,
  first_allele = c(
    NA, NA, "A001_B001", "A001_B002",
    NA, "A001_B002", "A001_B001"
  ),
  second_allele = c(
    NA, NA, "A010_B001", "A001_B001",
    NA, NA, NA
  ),
  stringsAsFactors = FALSE
)
pedWithGenotype <- addGenotype(ped, genotype)
pedGenotype <- getGVGenotype(pedWithGenotype)
allelesNewGen <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = pedGenotype,
  n = 5, updateProgress = NULL
)
```

---

getGVPopulation *Get the population of interest for the Genetic Value analysis*

Description

If user has limited the population of interest by defining pop, that information is incorporated via the ped$population column.

Usage

```
getGVPopulation(ped, pop)
```

Arguments

ped	The pedigree information in data.frame format
pop	character vector with animal IDs to consider as the population of interest. The default is NULL: an existing population column of ped is used if there is one, otherwise all animals.

Value

A logical vector with one element per row of ped, in the order of ped: TRUE if the animal is in the population of interest. IDs in pop that are not in ped are ignored.

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
ped$population <- getGVPopulation(ped, NULL)
```

---

getIdsWithOneParent *Get ids of animals with only one parent*

Description

Get ids of animals with only one parent

Usage

```
getIdsWithOneParent(uPed)
```

Arguments

uPed	a trimmed pedigree dataframe with uninformative founders removed.

Value

Character vector of the IDs of animals with exactly one known parent.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2, minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
probands <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
p <- addBackSecondParents(p, ped)
nrow(p)
getIdsWithOneParent(p)
```

---

getIncludeColumns *Get the superset of report-inclusion columns*

Description

Part of Genetic Value Functions

Usage

```
getIncludeColumns()
```

Details

Replaces INCLUDE.COLUMNS data statement.

Value

A character vector of the ten columns kept in genetic-value reports (`id`, `sex`, `age`, `birth`, `exit`, `population`, `condition`, `origin`, `first_name` and `second_name`). It is not the set of columns a pedigree file may contain; that is returned by `getPossibleCols`.

Examples

```
getIncludeColumns()
```

getLkDirectAncestors *Get the direct ancestors of selected animals*

Description

Gets direct ancestors from labkey study schema and demographics table.

Usage

```
getLkDirectAncestors(ids)
```

Arguments

ids	character vector of animal IDs

Value

data.frame with pedigree structure having the records of the Ids provided (the focal animals themselves) and all of their direct ancestors. NULL is returned when the site configuration cannot be read or the LabKey data cannot be fetched.

See Also

Other direct relatives: getFileDirectRelatives(), getLkDirectRelatives(), getPedDirectRelatives()

Examples

```
## Not run:
# Requires LabKey connection
library(nprcgenekeepr)
## Have to a vector of focal animals
focalAnimals <- c("1X2701", "1X0101")
suppressWarnings(getLkDirectAncestors(ids = focalAnimals))

## End(Not run)
```

getLkDirectRelatives *Get the direct relatives of selected animals from the LabKey EHR*

Description

Builds the pedigree of relatives for the provided focal animals from the LabKey study schema demographics table, obtained through the internal getPedigreeSource() adapter. The pedigree walk is delegated to getPedDirectRelatives(), so the result is the full connected pedigree component (ancestors, descendants, and collaterals such as siblings and mates) reachable from the focal animals.

Usage

```
getLkDirectRelatives(ids, unrelatedParents = FALSE)
```

Arguments

ids	character vector of animal IDs
unrelatedParents	logical vector when FALSE the unrelated parents of offspring do not get a record as an ego; when TRUE a place holder record where parent (sire, dam) IDs are set to NA.

Value

A data.frame with pedigree structure containing all direct relatives – the full connected pedigree component (ancestors, descendants, and collaterals) – for the Ids provided. NULL is returned when the LabKey data cannot be fetched.

See Also

Other direct relatives: getFileDirectRelatives(), getLkDirectAncestors(), getPedDirectRelatives()

Examples

```
## Not run:
# Requires LabKey connection
library(nprcgenekeepr)
## Have to a vector of focal animals
focalAnimals <- c("1X2701", "1X0101")
suppressWarnings(getLkDirectRelatives(ids = focalAnimals))

## End(Not run)
```

---

getOffspring *Get offspring to corresponding animal IDs provided*

Description

Get offspring to corresponding animal IDs provided

Usage

```
getOffspring(pedSourceDf, ids)
```

Arguments

pedSourceDf	dataframe with pedigree structure having at least the columns id, sire, and dam. The pedigree is the first argument, unlike getProbandPedigree(), getDescendantPedigree(), getPedDirectRelatives() and findOffspring(), which take the animal ids first.
ids	character vector of animal IDs

Value

A character vector containing all of the offspring IDs for all of the IDs provided in the second argument `ids`. All offspring are combined and duplicates are removed.

Examples

```
library(nprcgenekeepr)
pedOne <- nprcgenekeepr::pedOne
names(pedOne) <- c("id", "sire", "dam", "sex", "birth")
getOffspring(pedOne, c("s1", "d2"))
```

`getParents` *Get parents to corresponding animal IDs provided*

Description

Get parents to corresponding animal IDs provided

Usage

```
getParents(pedSourceDf, ids)
```

Arguments

`pedSourceDf`	dataframe with pedigree structure having at least the columns id, sire, and dam. The pedigree is the first argument, unlike `getProbandPedigree()`, `getDescendantPedigree()`, `getPedDirectRelatives()` and `findOffspring()`, which take the animal ids first.
`ids`	character vector of animal IDs

Value

A character vector with the IDs of the parents of the provided ID list.

Examples

```
library(nprcgenekeepr)
pedOne <- nprcgenekeepr::pedOne
names(pedOne) <- c("id", "sire", "dam", "sex", "birth")
getParents(pedOne, c("o1", "d4"))
```

`getPedDirectRelatives` *Get the direct relatives of selected animals from a pedigree*

Description

Gets the full connected pedigree component reachable from the selected animals in the supplied pedigree (`ped`). The result includes ancestors, descendants, and collaterals such as siblings and mates. A `NULL` `ped` returns `NULL`.

Usage

```
getPedDirectRelatives(ids, ped, unrelatedParents = FALSE)
```

Arguments

`ids`	character vector of animal IDs
`ped`	pedigree dataframe object that is used as the source of pedigree information.
`unrelatedParents`	logical vector when `FALSE` the unrelated parents of offspring do not get a record as an ego; when `TRUE` they get a place holder record as an ego in which the parent (`sire`, `dam`) IDs are set to `NA`. Place holder records are made for every collected ID that is absent from `ped$id`, including a selected ID that is not in the pedigree.

Value

A data.frame of pedigree records for the selected animals and the full connected pedigree component (ancestors, descendants, siblings and mates) in `ped`, or `NULL` when `ped` is `NULL`.

See Also

Other direct relatives: `getFileDirectRelatives()`, `getLkDirectAncestors()`, `getLkDirectRelatives()`

Examples

```
library(nprcgenekeepr)
## A pedigree to search and a focal animal whose direct relatives we want
ped <- nprcgenekeepr::lacy1989Ped
getPedDirectRelatives(ids = "E", ped = ped)
```

getPedigree *Get pedigree from file*

Description

Get pedigree from file

Usage

```
getPedigree(fileName, sep = ",")
```

Arguments

fileName	character vector of temporary file path.
sep	column separator in CSV file. It is ignored for Excel (xls and xlsx) files.

Value

A data.frame of the pedigree as read from the file, with no quality control applied (see qcStudbook). Every column read from an Excel file is returned as character.

Examples

```
library(nprcgenekeepr)
ped <- getPedigree(fileName = system.file("testdata", "qcPed.csv",
  package = "nprcgenekeepr"
))
```

getPedMaxAge *Get the maximum age of any animal in the pedigree*

Description

Returns the maximum age among all animals in the pedigree that have a non-NA age. Because ages are computed for deceased animals (age at exit) as well, the maximum can reflect a deceased animal.

Usage

```
getPedMaxAge(ped)
```

Arguments

ped	The pedigree information in data.frame format

Value

Numeric value representing the maximum age of animals in the pedigree, or NA_real_ if no animal has a non-missing age (no age column or all ages missing).

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
ped <- qcStudbook(examplePedigree,
  minSireAge = 2, minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
getPedMaxAge(ped)
```

---

getPossibleCols *Get possible column names for a studbook*

Description

Pedigree curation function

Usage

```
getPossibleCols()
```

Value

A character vector of the possible columns that can be in a studbook. The possible columns are as follows:

id	– character vector with unique identifier for an individual
sire	– character vector with unique identifier for an individual's father (NA if unknown).
dam	– character vector with unique identifier for an individual's mother (NA if unknown).
sex	– factor (levels: "M", "F", "U") Sex specifier for an individual
species	– character vector or NA (optional) naming the species of the individual (e.g. "rhesus"). Recognized as a first-class column rather than retained as an unrecognized novel column.
gen	– integer vector with the generation number of the individual
birth	– Date or NA with the individual's birth date. This is one of the required columns (see getRequiredCols); the date value itself may be NA if unknown.
exit	– Date or NA (optional) with the individual's exit date (death, or departure if applicable)

ancestry	– character vector or NA (optional) that indicates the geographic population to which the individual belongs.
age	– numeric or NA (optional) indicating the individual's current age or age at exit.
population	– an optional logical argument indicating whether or not the id is part of the extant population.
origin	– character vector or NA (optional) that indicates the name of the facility that the individual was imported from. NA indicates the individual was not imported.
status	– an optional factor indicating the status of an individual with levels ALIVE, DECEASED, SHIPPED, and UNKNOWN.
condition	– character vector or NA (optional) that indicates the restricted status of an animal. "Nonrestricted" animals are generally assumed to be naive.
spf	– character vector or NA (optional) indicating the specific pathogen-free status of an individual.
vasxOvx	– character vector indicating the vasectomy/ovariectomy status of an animal where NA indicates an intact animal and all other values indicate surgical alteration.
death	– Date or NA (optional) with the individual's death date.
departure	– Date or NA (optional) with the individual's departure date.
first	– character vector or NA (optional) with the first value of a genotype pair. The name is reserved; genotype files may not use it.
second	– character vector or NA (optional) with the second value of a genotype pair. The name is reserved; genotype files may not use it.
first_name	– character vector or NA (optional) with the first value of a genotype pair in the single-locus genotype format.
second_name	– character vector or NA (optional) with the second value of a genotype pair in the single-locus genotype format.
recordStatus	– character vector with value "original" for records that were in the input and "added" for records added for parents that had no record of their own.
pedNum	– integer vector indicating generation numbers for each id, starting at 0 for individuals lacking IDs for both parents.
affected	– logical vector or NA (optional) indicating whether an individual is affected by a condition of interest, shaded on the Pedigree Diagram tab (issue #133). Matches kinship2's own affected argument naming/semantics; NA means unknown affected status, not "not affected".
name	– character vector or NA (optional) giving a human-readable animal name/nickname, shown alongside id on the Pedigree Diagram tab when enabled (issue #136). NA or an empty string falls back to id; not all centers record this field, so its presence and completeness both vary by animal. Scrubbed to NA by obfuscatePed.

Examples

```
library(nprcgenekeepr)
getPossibleCols()
```

`getPotentialParents` *Get potential parents for animals with unknown parents*

Description

[Experimental]

Usage

```
getPotentialParents(
  ped,
  minSireAge = NULL,
  minDamAge = NULL,
  minParentAge = lifecycle::deprecated(),
  maxGestationalPeriod = NULL,
  gestationTable = NULL
)
```

Arguments

`ped`	the pedigree information in data.frame format. Pedigree (req. fields: id, sire, dam, sex, birth, exit, fromCenter; the `species` field is optional). The `gen` and `population` fields are not used. This requires complete pedigree information.
`minSireAge`	numeric minimum age in years for a male to be proposed as a potential sire. `NULL` (default) looks up the floor for each candidate's species via `getSpeciesMinBreedingAge` (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor for all male candidates. Candidates with missing birth dates are not considered.
`minDamAge`	numeric minimum age in years for a female to be proposed as a potential dam. `NULL` (default) looks up the floor for each candidate's species via `getSpeciesMinBreedingAge` (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor for all female candidates.
`minParentAge`	**[Deprecated]** Deprecated scalar minimum parent age. Supplying it sets both `minSireAge` and `minDamAge`; use those sex-specific parameters instead. Supplying `NULL` disables the age check entirely.
`maxGestationalPeriod`	integer maximum number of days between conception and birth for the species being analyzed (a conservative upper bound, e.g. 210 for rhesus whose typical gestation is about 165 days). When `NULL` (the default) the window is looked up per animal from the `species` column of `ped` via `getSpeciesGestation`, falling back to 210 days for animals whose species is missing or unrecognized; supply an explicit integer to use one fixed window for every animal. It is used two ways: (1) a sire who exited the colony between conception (birth - maxGestationalPeriod) and birth is still retained as a candidate; and (2) a female who delivered another offspring within maxGestationalPeriod days of the focal birth is excluded as a candidate dam, because a female bears one offspring at a time.

The sire check uses presence at conception while the dam check uses presence at birth; this asymmetry is intentional – a sire need only be present to conceive, whereas a dam must be present through the pregnancy to give birth.

`gestationTable` optional data.frame (columns `species`, `gestation`) passed to `getSpeciesGestation` for the per-animal lookup when `maxGestationalPeriod` is `NULL`. Defaults to `NULL`, which uses the bundled `speciesGestation` table.

Value

a list of list with each internal list being made up of an animal id (`id`), a vector of possible sires (`sires`) and a vector of possible dams (`dams`). The `id` must be defined while the vectors `sires` and `dams` can be empty. A fourth element, `damBasis`, is a single string saying where `dams` came from: `"provenBreeder"` when the candidates are females who gave birth near the time of the focal birth, `"eligibleFemale"` when there were none and `dams` lists every female old enough and present at the birth instead, and `NA` when `dams` is empty. In both tiers, a female who delivered another offspring within `maxGestationalPeriod` days of the focal birth is never listed. Candidates are listed only for the parent that is missing: when an animal's dam is recorded its `dams` is empty, and when its sire is recorded its `sires` is empty. `NULL` (not a list) is returned when no animal has an unknown parent with a candidate and when `ped` has no `fromCenter` column.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::rhesusPedigree
## getPotentialParents needs a logical fromCenter column flagging
## colony-born animals; add one if your pedigree lacks it.
ped$fromCenter <- TRUE
potentialParents <- getPotentialParents(
  ped = ped, minSireAge = 2, minDamAge = 2, maxGestationalPeriod = 210L
)
## Each element pairs a focal id with candidate sires and dams.
potentialParents[[1L]]
```

`getPotentialSires` *List potential sires*

Description

List potential sires

Usage

```
getPotentialSires(ids, ped, minAge = 1L)
```

Arguments

ids	character vector of animal IDs
ped	dataframe that is the Pedigree. It contains pedigree information including the IDs listed in ids.
minAge	integer value giving the inclusive minimum current age (in years) a male must have to be listed as a potential sire. Default is 1 year.

Value

A character vector of potential sire Ids

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedWithGenotype
ids <- nprcgenekeepr::qcBreeders
getPotentialSires(ids, ped, minAge = 1L)
```

---

getProbandPedigree *Reduce a pedigree to probands and their ancestors*

Description

Filters a pedigree down to the provided group (the probands) together with all of their ancestors, removing unnecessary individuals from the studbook. This version builds the pedigree back in time starting from a group of probands. This will include all ancestors of the probands, even ones that might be uninformative.

Usage

```
getProbandPedigree(probands, ped)
```

Arguments

probands	a character vector with the list of animals whose ancestors should be included in the final pedigree.
ped	datatable that is the Pedigree. It contains pedigree information. The fields id, sire and dam are required.

Value

A reduced pedigree.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedWithGenotype
ids <- nprcgenekeepr::qcBreeders
sires <- getPotentialSires(ids, ped, minAge = 1)
head(getProbandPedigree(probands = sires, ped = ped))
```

---

getPyramidAgeDist *Get the age distribution for the pedigree*

Description

Returns the pedigree with all animals, adding a `status` column describing each animal as `ALIVE` or `DECEASED` and a computed `age` column (age at exit for deceased animals). All animals are returned, not only living ones. An animal is `DECEASED` when it has an exit date or when its birth date is `NA`; otherwise it is `ALIVE`. When `ped` is `NULL` (the default), the packaged `qcPed` example data are used.

Usage

```
getPyramidAgeDist(ped = NULL)
```

Arguments

`ped`	data.frame whose first six columns are id, sire, dam, sex, birth and exit, in that order (see Details above). Defaults to `NULL`, which uses the packaged `qcPed` example data.

Details

The columns of `ped` are read by position, not by name: the first six columns must be the id, sire, dam, sex, birth and exit values in that order, and are renamed accordingly. Any further columns are dropped. If birth is not in the fifth column and of class `Date`, `POSIXct` or `character`, the error message reports that the birth column is of the wrong class.

The lubridate package is used here because of the way the modern Gregorian calendar is constructed, there is no straightforward arithmetic method that produces a person's age, stated according to common usage — common usage meaning that a person's age should always be an integer that increases exactly on a birthday.

Value

A pedigree with the six columns of `ped` (`id`, `sire`, `dam`, `sex`, `birth`, `exit`) and a `status` column added, which describes the animal as `ALIVE` or `DECEASED` and a `age` column added, which has the animal's age in years or `NA` if it cannot be calculated. The `exit` column values have been remapped to valid dates or `NA`.

Examples

```
library(nprcgenekeepr)
ped <- getPyramidAgeDist()
```

`getPyramidPlot` *Create an age-sex pyramid plot of a pedigree*

Description

The pedigree provided must have the following columns: `sex` and `age`. This needs to be augmented to allow pedigrees structures that are provided by the nprcgenekeepr package.

Usage

```
getPyramidPlot(
  ped = NULL,
  binWidth = 2L,
  ageUnit = "years",
  colorScheme = "default",
  showCounts = TRUE,
  ageLabelCex = 1
)
```

Arguments

`ped`	The pedigree information in data.frame format
`binWidth`	numeric bin width for age groups (default 2). The value is truncated to a whole number and is at least 1, so 0.5 gives 1 and 2.9 gives 2.
`ageUnit`	character either "years" (default) or "months".
`colorScheme`	character color scheme: "default" (blue/pink) or "viridis" (colorblind-friendly).
`showCounts`	logical whether to show count values on bars (default TRUE).
`ageLabelCex`	numeric expansion factor for age labels (default 1.0).

Details

Only living animals are plotted: when `ped` has an `exit` column, animals with a non-`NA` `exit` are dropped, and only animals with sex "M" or "F" are counted. The age axis is, however, sized from the oldest animal in `ped`, including deceased ones. When `ped` is `NULL` (the default), the packaged `qcPed` example data are used. An unrecognized `colorScheme` silently falls back to "default".

Value

The return value of par("mar") when the function was called.

Examples

```
library(nprcgenekeepr)
data(qcPed)
getPyramidPlot(qcPed)
getPyramidPlot(qcPed, binWidth = 5, colorScheme = "viridis")
```

getRequiredCols *Get required column names for a studbook*

Description

Pedigree curation function

Usage

```
getRequiredCols()
```

Value

A character vector of the required columns that can be in a studbook. The required columns are as follows:

- `id` – character vector with unique identifier for an individual
- `sire` – character vector with unique identifier for an individual's father (`NA` if unknown).
- `dam` – character vector with unique identifier for an individual's mother (`NA` if unknown).
- `sex` – factor (levels: "M", "F", "U") Sex specifier for an individual
- `birth` – Date or `NA` with the individual's birth date

Examples

```
library(nprcgenekeepr)
getRequiredCols()
```

getSiteInfo *Get site information*

Description

Get site information

Usage

```
getSiteInfo(expectConfigFile = TRUE)
```

Arguments

`expectConfigFile`
: logical parameter when set to `FALSE`, no configuration is looked for. Default value is `TRUE`.

Value

A named list of 20 elements of site specific information used by the application.

The first seven elements (`center`, `baseUrl`, `schemaName`, `folderPath`, `queryName`, `lkPedColumns` and `mapPedColumns`) are read from the configuration file when one exists. A configuration file that lacks one of these keys causes an error ("Could not find ..."). When no configuration file exists, the defaults for the ONPRC are returned instead: center "ONPRC", baseUrl `"https://primeuat.ohsu.edu"`, schemaName "study", folderPath "/ONPRC/EHR" and queryName "demographics". A warning is signaled for the missing file only when `expectConfigFile` is `TRUE`.

The format of the configuration file is described in the notes at the top of the example file, `example_nprcgenekeepr_config`, in the package's `extdata/examples` folder.

The returned list contains the following elements.

1. `center` – center name, such as "ONPRC" or "SNPRC"
2. `baseUrl` – base URL of the LabKey server
3. `schemaName` – LabKey schema name
4. `folderPath` – LabKey folder path
5. `queryName` – LabKey query name, "demographics" by default
6. `lkPedColumns` – LabKey column names for the pedigree
7. `mapPedColumns` – the package column names that `lkPedColumns` are renamed to
8. `sysname`, `release`, `version`, `nodename`, `machine`, `login`, `user` and `effective_user` – character strings from `Sys.info()`
9. `homeDir` and `configFile` – the home directory and the expected configuration file path, from `getConfigFileName`
10. `requiredCols` – the required studbook columns, from `getRequiredCols`
11. `possibleCols` – the possible studbook columns, from `getPossibleCols`
12. `includeColumns` – the superset of report-inclusion columns, from `getIncludeColumns`

Examples

```
library(nprcgenekeepr)
## default sends warning if configuration file is missing
suppressWarnings(getSiteInfo())
getSiteInfo(expectConfigFile = FALSE)
```

`getSpeciesGestation` *Look up the maximum gestation period (days) for one or more species*

Description

Maps each supplied species name to a conservative upper bound on the number of days from conception to birth, using the `speciesGestation` lookup table (or a supplied `gestationTable`). Matching is case- and whitespace-insensitive. Any species that is missing, `NA`, an empty string, or not present in the table falls back to `default` (210 days, the conservative rhesus bound). Used by `getPotentialParents` to key its gestation window on the first-class `species` pedigree column.

Usage

```
getSpeciesGestation(species, gestationTable = NULL, default = 210L)
```

Arguments

species	character vector of species names (may contain NA).
gestationTable	optional data.frame with a character column species and an integer column gestation to use instead of the bundled speciesGestation table. Defaults to NULL, which uses the bundled table.
default	integer fallback returned for species that are missing, NA, empty, or not found in the table. Defaults to 210L.

Value

an integer vector of gestation-period day bounds, the same length and order as species.

Examples

```
getSpeciesGestation("RHESUS")
getSpeciesGestation(c("RHESUS", "UNICORN", NA))
```

---

getSpeciesMinBreedingAge

Look up the minimum breeding age (years) for one or more species and sexes

---

Description

Maps each supplied (species, sex) pair to the minimum age in years at which an animal of that species and sex can produce offspring, using the speciesGestation lookup table (or a supplied breedingTable). Matching is case- and whitespace-insensitive on both species and sex. Any species that is missing, NA, an empty string, or not present in the table – and any sex that is not "M" or "F" – falls back to default (2 years, the legacy package-wide minimum parent age). Used by the Genetic Value Analysis unknown-parent mean-kinship correction to form a focal animal's contemporaneous breeding-age peer cohort. The bundled table is populated for the common colony NHP species; a user's overrides, merged onto the bundled table by loadSpeciesOverrides, are supplied through breedingTable and default.

Usage

```
getSpeciesMinBreedingAge(species, sex, breedingTable = NULL, default = 2)
```

Arguments

species	character vector of species names (may contain NA).
sex	character vector of sexes ("M" or "F"); recycled to the length of species (or vice versa).
breedingTable	optional data.frame with a character column species and numeric columns minMaleBreedingAge and minFemaleBreedingAge to use instead of the bundled speciesGestation table. Defaults to NULL, which uses the bundled table.
default	numeric fallback returned for species that are missing, NA, empty, or not found, and for a sex that is not "M"/"F". Defaults to 2.

Value

a numeric vector of minimum breeding ages in years, the same length as the longer of species and sex.

Examples

```
getSpeciesMinBreedingAge("RHESUS", "M")
getSpeciesMinBreedingAge("RHESUS", "F")
getSpeciesMinBreedingAge(c("RHESUS", "UNICORN"), c("M", "F"))
```

---

getTokenList *Get tokens from a character vector of lines*

Description

Reads the lines of an nprcgenekeepr configuration file. The term before each equals sign names a setting and the terms after it, up to the next name, are its value; terms are separated by blanks, commas or line breaks, a term in double quotes is kept whole (a quote not closed on its line ends at the end of that line), and a # that begins a word starts a comment that runs to the end of the line. Round brackets outside quotes are ignored. A setting with nothing after its equals sign has an empty value.

Usage

```
getTokenList(lines)
```

Arguments

lines	character vector with text from configuration file

Value

A list with two elements: param, a character vector of parameter names, and tokenVec, a list of the token vectors parsed for each parameter (an empty character vector for a setting with no value).

Examples

```
lines <- c(
  "center = \"SNPRC\"",
  " baseUrl = \"https://boomer.txbiomed.local:8080/labkey\"",
  " schemaName = \"study\"", " folderPath = \"/SNPRC\"",
  " queryName = \"demographics\"",
  "lkPedColumns = (\"Id\", \"gender\", \"birth\", \"death\",",
  "               \"lastDayAtCenter\", \"dam\", \"sire\")",
  "mapPedColumns = (\"id\", \"sex\", \"birth\", \"death\", ",
  "  \"exit\", \"dam\", \"sire\")"
)
lkVec <- c(
  "Id", "gender", "birth", "death",
  "lastDayAtCenter", "dam", "sire"
)
mapVec <- c("id", "sex", "birth", "death", "exit", "dam", "sire")
tokenList <- getTokenList(lines)
params <- tokenList$param
tokenVectors <- tokenList$tokenVec
```

`getVersion` *Get the version number of nprcgenekeepr*

Description

Get the version number of nprcgenekeepr

Usage

```
getVersion(date = TRUE)
```

Arguments

`date`	A logical value when TRUE (default) a date in YYYY-MM-DD (ISO) format within parentheses is appended.

Value

Current Version

Examples

```
library(nprcgenekeepr)
getVersion()
```

get_and_or_list *Join a character vector into an and/or list*

Description

Join a character vector into an and/or list

Usage

```
get_and_or_list(c_vector, conjunction = "and")
```

Arguments

c_vector	Character vector containing the list of words to be put in a list.
conjunction	The conjunction to be used as the connector. This is usually 'and' or 'or' with 'and' being the default.

Value

A character vector of length one containing a single correctly punctuated character string that lists each element in the first argument vector with commas between if there are more than two elements with the last two elements joined by the selected conjunction. A zero-length input returns an empty string and a single element is returned unchanged.

Examples

```
get_and_or_list(c("Bob", "John")) # "Bob and John"
get_and_or_list(c("Bob", "John"), "or") # "Bob or John"
get_and_or_list(c("Bob", "John", "Sam", "Bill"), "or")
# "Bob, John, Sam, or Bill"
```

get_elapsed_time_str *Format the elapsed time since a start time*

Description

Taken from github.com/rmsharp/rmsutilityr

Usage

```
get_elapsed_time_str(start_time)
```

Arguments

start_time	a proc_time object as returned by proc.time()

Value

A character vector describing the passage of time in hours, minutes, and seconds.

Examples

```
start_time <- proc.time()
## do something
elapsed_time <- get_elapsed_time_str(start_time)
```

`groupAddAssign` *Add animals to a breeding group or form new groups*

Description

Part of Group Formation

Usage

```
groupAddAssign(
  candidates,
  kmat,
  ped,
  currentGroups = list(character(0L)),
  threshold = 0.015625,
  ignore = list(c("F", "F")),
  minAge = 1,
  iter = 1000L,
  numGp = 1L,
  harem = FALSE,
  sexRatio = 0,
  withKin = FALSE,
  maxCandidates = 5L,
  exhaustive = FALSE,
  maxExhaustiveCandidates = 20L,
  exhaustiveTimeLimit = 10,
  ancestryRules = NULL,
  updateProgress = NULL
)
```

Arguments

`candidates`	Character vector of IDs of the animals available for use in forming the groups. The animals that may be present in `currentGroups` are not included within `candidates`. Ids given as numbers or as a factor, here, in `currentGroups` and in the pedigree's `id`, `sire` and `dam`, are taken as their text.
`kmat`	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.

ped	dataframe that is the Pedigree. It contains pedigree information including the IDs listed in ids.
currentGroups	List of character vectors of IDs of animals currently assigned to groups. Defaults to list(character(0L)), a list with a single empty element regardless of numGp, assuming no groups are prepopulated.
threshold	Numeric value indicating the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored. The default value is 0.015625.
ignore	List of character vectors representing the sex combinations to be ignored. If provided, the vectors in the list specify if pairwise kinship should be ignored between certain sexes. Default is to ignore all pairwise kinship between females.
minAge	Integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal younger than this age (by the age column of ped) are ignored; an animal exactly this old is not ignored. Default is 1 year.
iter	Integer indicating the number of times to perform the random group formation process. Default value is 1000 iterations.
numGp	Integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
harem	Logical variable when set to TRUE, the formed groups have a single male at least minAge old. A harem's sire is held to the same rules as every other animal: animals related to him at or above threshold (and, with ancestryRules, animals a block rule bars from him) are not placed in his group, and when that leaves no one the group is the sire alone. A group whose seed animals (currentGroups) include a male at least minAge old keeps him as its sire; a group with no seeded male is given a sire chosen at random from candidates. It cannot be combined with a non-zero sexRatio; the call stops with a message naming both settings.
sexRatio	Numeric value indicating the number of non-males (females and animals of other or unknown sex) per male. Values from 0.5 to 20 in steps of 0.5 are typical, but the range is not enforced. A male is placed only if he is at least minAge years old; a male younger than that, or with no birth date, is left out of the groups. A group stops being filled when it has no such male left, so it can end with fewer females than the ratio asks for. A non-zero sexRatio cannot be combined with harem = TRUE; the call stops with a message.
withKin	Logical variable when set to TRUE, the kinship matrix for the group is returned along with the group and score. Defaults to not return the kinship matrix. This maintains compatibility with earlier versions.
maxCandidates	Integer value indicating the maximum number of distinct candidate solutions to retain during the simulation (issue #146 Slice 1). Default is 5.
exhaustive	Logical. When TRUE, enumerate every maximal independent set of the conflict graph instead of random sampling (issue #146 Slice 2), subject to maxExhaustiveCandidates/ exhaustiveTimeLimit. Only supported for numGp = 1, harem = FALSE, sexRatio = 0 – any other combination stop()s with a message naming the specific unsupported condition, rather than silently falling back to sampling. Default is FALSE (the existing sampling behavior, unchanged).

maxExhaustiveCandidates
: Integer. Pre-flight feasibility ceiling for `exhaustive = TRUE`: a candidate pool larger than this `stop()`s before any enumeration runs. Default is 20.

exhaustiveTimeLimit
: Numeric. Wall-clock seconds allowed for `exhaustive = TRUE`'s search before it truncates gracefully (`exhaustive = FALSE` in the return value, not an error). Default is 10.

ancestryRules
: Optional data frame of ancestry compatibility rules (issue #168), in the shape `checkAncestryRules` validates – it is (re-)validated here, so structural or domain errors `stop()` with the validator's specific message. Default `NULL`: no rules, behavior identical to previous versions. When supplied, `ped` must carry an `ancestry` column (an error otherwise, so a script user who passed rules gets truth, not silence). `block`-severity rules exclude their conflicting pairs during formation – in sampling, exhaustive, and `sexRatio` modes, and sex-blind (deliberately bypassing the female-female kinship exemption in `ignore`). `flag`-severity rules never affect the search; compute them afterward with `reportAncestryViolations`.

updateProgress
: Function or NULL. If this function is defined, it will be called during each iteration to update a `shiny::Progress` object.

Details

`groupAddAssign` finds the largest group that can be formed by adding unrelated animals from a set of candidate IDs to an existing group, to a new group it has formed from a set of candidate IDs or if more than 1 group is desired, it finds the set of groups with the largest minimum group size (the score is the size of the smallest group).

The function implements a maximal independent set (MIS) algorithm to find groups of unrelated animals. A set of animals may have many different MISs of varying sizes, and finding the largest would require traversing all possible combinations of animals. Since this could be very time consuming, this algorithm produces a random sample of the possible MISs, and selects from these. The size of the random sample is determined by the specified number of iterations.

Value

A list with list items `group`, `score`, `candidates` and optionally `groupKin`. The list item `group` contains a list of the best group(s) produced during the simulation (an alias for `candidates[[1]]$group`, kept for backward compatibility). It has `numGp` groups plus a final element holding the IDs of the unused candidates, or a bare `NA` if none remain. The same holds for each `candidates[[i]]$group`. The list item `score` provides the score associated with the group(s) (an alias for `candidates[[1]]$score`); it ignores that final element. The list item `candidates` is a list of up to `maxCandidates` distinct candidate solutions (default 5; issue #125, configurable per issue #146 Slice 1), each a list with its own `group`, `score` and, when `withKin = TRUE`, `groupKin`, ordered best-scoring first. Candidates are deduplicated by partition content, not by score – two trials with the same score but different membership both count as distinct candidates; two trials with identical membership count once. The list item `groupKin` contains the subset of the kinship matrix that is specific for each group formed in the best candidate (an alias for `candidates[[1]]$groupKin`). When `exhaustive = TRUE` was requested (issue #146 Slice 2), three additional top-level items are present – absent entirely, not merely `NULL`, when `exhaustive = FALSE` (the default): `exhaustive` (logical, whether the search completed before its wall-clock deadline), `examined` (integer, the total number of distinct maximal

independent sets found), and `retentionRule` (character, describing the top-`maxCandidates` cutoff actually applied to `candidates`).

References

Vinson, A. and Raboin, M.J. (2015) "A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques (*Macaca mulatta*)" *Journal of the American Association for Laboratory Animal Science*, 2015 Nov, Vol.54(6), pp.700-707.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2, minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped,
  removeUninformative = FALSE,
  addBackParents = FALSE
)
geneticValue <- reportGV(ped,
  guIter = 50, # should be >= 1000
  guThresh = 3,
  byID = TRUE,
  updateProgress = NULL
)
trimmedGeneticValue <- reportGV(trimmedPed,
  guIter = 50, # should be >= 1000
  guThresh = 3,
  byID = TRUE,
  updateProgress = NULL
)
candidates <- trimmedPed$id[trimmedPed$birth < as.Date("2013-01-01") &
  !is.na(trimmedPed$birth) &
  is.na(trimmedPed$exit)]
haremGrp <- groupAddAssign(
  kmat = trimmedGeneticValue[["kinship"]],
  ped = trimmedPed,
  candidates = candidates,
  iter = 10, # should be >= 1000
  numGp = 6,
  harem = TRUE
)
```

```
haremGrp$group
sexRatioGrp <- groupAddAssign(
  kmat = trimmedGeneticValue[["kinship"]],
  ped = trimmedPed,
  candidates = candidates,
  iter = 10L, # should be >= 1000L
  numGp = 6L,
  sexRatio = 9.0
)
sexRatioGrp$group
```

---

gvaConvergence *Recommend gene-drop iterations for a pedigree*

Description

Part of Genetic Value Analysis

Usage

```
gvaConvergence(
  ped,
  pop = NULL,
  nMax = 3000L,
  guThresh = 1L,
  byID = TRUE,
  grid = NULL,
  k = 20L,
  oMin = 0.9,
  rhoMin = 0.95,
  seed = NULL,
  updateProgress = NULL,
  breedingTable = NULL,
  gestationTable = NULL,
  breedingAgeDefault = NULL,
  gestationDefault = NULL,
  kinshipOverrides = NULL,
  twinRelations = NULL
)
```

Arguments

ped	The pedigree information in data.frame format (the same input reportGV takes).
pop	Character vector with animal IDs to consider as the population of interest. The default is NULL: an existing population column of ped is used if there is one, otherwise all animals.

nMax — Integer gene-drop budget: the number of iteration columns to simulate. Reproducibility is assessed for iteration counts N with 2 * N <= nMax (each half-split needs 2 * N columns). Default 3000.

guThresh — Integer threshold number of animals for defining a rare (unique) allele, passed to calcA. Default 1.

byID — Logical passed to alleleFreq() via calcA; if TRUE, homozygous alleles are counted once per individual. Default TRUE.

grid — Integer vector of candidate iteration counts to assess. The default builds c(25, 50, 100, 200, 400, 800, 1500), keeping only those with 2 * N <= nMax.

k — Integer size of the top-k selected set compared for overlap. Default 20.

oMin — Numeric minimum top-k overlap for reproducibility. Default 0.90.

rhoMin — Numeric minimum Kendall rank agreement for reproducibility. Default 0.95.

seed — Optional integer; when supplied, set_seed pins the gene-drop RNG so the convergence curve is reproducible. Default NULL.

updateProgress — Function or NULL passed through to geneDrop() to update a shiny::Progress object; it is called once at the start and then once for each animal processed. Default NULL.

breedingTable, gestationTable, breedingAgeDefault, gestationDefault — Optional overrides for the unknown-parent mean-kinship correction, passed through to correctUnknownParentMeanKinship() exactly as reportGV passes them. NULL uses the bundled defaults.

kinshipOverrides — Optional data.frame of outside-information kinship overrides (id1, id2, kinship; the coefficient *f*, not relatedness *r*) applied to the kinship matrix before mean kinship and the unknown-parent correction, exactly as reportGV applies them, so the convergence diagnostic ranks on the same mean kinship the report uses. NULL (the default) leaves the pedigree-derived matrix unchanged. Ids outside the analysis set are warn-dropped (the run is not aborted). An override REFINES the named kinship cell; it does not suppress the + sexMean / 2 unknown-parent correction, which is kept for every animal missing one parent. See applyKinshipOverrides.

twinRelations — Optional data.frame of declared twin pairs (id1, id2, code) passed straight through to the internal kinship call, exactly as reportGV passes it. Only code == "MZ twin" rows affect the computed matrix. NULL (the default) is a complete no-op, matching today's behavior.

Details

Genome uniqueness (calcGU) is the only ranked Genetic Value Analysis output that carries Monte Carlo (gene-drop) sampling noise, so the number of iterations a colony actually needs is pedigree-dependent: there is no single universal "right" count. gvaConvergence answers the literal request – "define reproducible and automate finding the needed number of iterations" – on the ratified definition that the decision-relevant quantity is the *selection order* (which animals are chosen, and in what order), not the precision of the gu number itself.

Because the n gene-drop iteration columns are independent and identically distributed replicates, the whole convergence picture is recoverable from a *single* completed gene drop the user already pays for: gvaConvergence runs one gene drop at nMax, computes the per-iteration rare-allele matrix

once (calcA), and for each candidate iteration count N in grid splits the columns into two disjoint halves of N columns each. The two halves are genuinely independent N-iteration estimates; each is ranked through the same ordering pipeline the report uses, and the two orderings are compared. A run is judged **reproducible at** N when both

- the top-k selected animals overlap by at least oMin (the same animals are chosen), and
- the Kendall rank agreement of the commonly-ranked animals is at least rhoMin (they come out in the same order).

The recommended iteration count is the smallest N in grid at which both criteria hold. The de-inflated gu = 0 "Undetermined" set (both parents unknown, no recorded origin) is a policy constant with rank NA; it is excluded from the order the criteria are computed on and reported separately as nUndetermined.

Because the half-split compares two N-column runs to *each other* (never to their pooled mean), it is a conservative, self-validating estimate of reproducibility at N. This changes nothing about seeding: a fixed seed already makes gu bit-identical run to run; that is reproducibility of the *process*, whereas this function reports the sampling reproducibility of the *estimate*.

The population of interest (see pop) must hold at least 2 animals, as for reportGV. With fewer, gvaConvergence stops before doing any calculation, with a message giving the number of animals.

Value

An object of class nprcgenekeeprGVConv: a list with

- convergence – a data.frame with one row per assessed iteration count: iterations, topOverlap (top-k selected-set overlap, from 0 to 1), and rankAgreement (Kendall rank agreement of the commonly-ranked animals, from -1 to 1). Either metric can be NA when it cannot be computed.
- recommendedIter – the smallest assessed iteration count meeting both criteria, or NA_integer_ if none did within grid.
- converged – TRUE if any assessed count met both criteria.
- criteria – the k, oMin, and rhoMin used.
- nRankable – the number of probands carrying a (non-NA) rank that the order metrics are computed on.
- nUndetermined – the count of the excluded Undetermined set.
- nMax – the gene-drop budget actually simulated.

See Also

reportGV, calcGU, calcGUSE

Examples

```
library(nprcgenekeepr)
## A quick, small illustration (use a larger nMax in practice). With
## nMax = 200 only the grid counts 25, 50 and 100 can be assessed.
conv <- gvaConvergence(nprcgenekeepr::qcPed, nMax = 200L, seed = 1L)
conv$convergence
conv$recommendedIter
```

hasBothParents *Check whether an animal has both parents*

Description

Check whether an animal has both parents

Usage

```
hasBothParents(id, ped)
```

Arguments

id	a single ID to examine for parents. A vector of IDs is not supported: the result is not one value per ID, and a recycling warning appears only when the number of IDs does not divide the number of rows of ped.
ped	The pedigree information in data.frame format

Value

TRUE if ID has both sire and dam identified in ped, FALSE if one or both are unknown, and `logical(0)` if id is not in ped.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedOne
names(ped) <- c("id", "sire", "dam", "sex", "birth")
hasBothParents("o2", ped)
ped$sire[ped$id == "o2"] <- NA
hasBothParents("o2", ped)
```

hasGenotype *Check for genotype data in dataframe*

Description

Checks to ensure the content and structure are appropriate for genotype data are in the dataframe and ready for the geneDrop function by already being mapped to numbers (integer codes are expected, but any numeric value passes) and placed in columns named first and second. Only the id column is matched loosely (any column whose lower-cased name contains "id"). The allele columns must be named exactly first and second and be numeric; no check is made of the values' range, so negative, missing or very large numbers still give TRUE.

Usage

```
hasGenotype(genotype)
```

Arguments

genotype	dataframe with genotype data

Value

A logical value representing whether or not the data.frame passed in contains genotypic data that can be used. Columns named First or Second (other cases) give FALSE.

Examples

```
library(nprcgenekeepr)
rhesusPedigree <- nprcgenekeepr::rhesusPedigree
rhesusGenotypes <- nprcgenekeepr::rhesusGenotypes
pedWithGenotypes <- addGenotype(
  ped = rhesusPedigree,
  genotype = rhesusGenotypes
)
hasGenotype(pedWithGenotypes)
```

---

headerDisplayNames *Convert internal column names to display or header names*

Description

Converts the column names of a Pedigree or Genetic value Report to something more descriptive.

Usage

```
headerDisplayNames(headers)
```

Arguments

headers	a character vector of column (header) names

Value

A character vector the same length as headers with the display name for each header. The lookup is case sensitive, and a header with no display name gives NA.

Examples

```
library(nprcgenekeepr)
headerDisplayNames(headers = c("id", "sire", "dam", "sex", "birth", "age"))
```

`isFounder` *Identify the founders in a pedigree*

Description

Part of Pedigree Curation

Usage

```
isFounder(ped)
```

Arguments

`ped`	a pedigree `data.frame` with (at least) the columns `sire` and `dam`.

Details

A founder is an animal whose sire and dam are both unknown (`NA`). Animals with exactly one known parent (partial parentage) are **not** founders.

Value

A logical vector with one element per row of `ped` that is `TRUE` for each animal whose sire and dam are both `NA`. The result never contains `NA`.

See Also

`getFounders` for the founder `id` values.

Examples

```
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
isFounder(ped)
```

is_valid_date_str *Test whether a string is a valid date*

Description

Taken from github.com/rmsharp/rmsutilityr

Usage

```
is_valid_date_str(
  date_str,
  format = "%d-%m-%Y %H:%M:%S",
  optional = FALSE
)
```

Arguments

date_str	character vector with 0 or more dates
format	character vector of length one. Retained for backward compatibility; the current implementation validates dates with anytime() and does not use format.
optional	logical value indicating that NA should be returned instead of FALSE for strings that are not valid dates. Defaults to FALSE.

Value

A logical vector with one element per element of date_str (zero-length input gives a zero-length result) indicating whether or not each string is a valid date. Elements are NA instead of FALSE when optional is TRUE. Numeric input is never treated as a date and always gives FALSE.

Examples

```
is_valid_date_str(c(
  "13-21-1995", "20-13-98", "5-28-1014",
  "1-21-15", "2-13-2098", "25-28-2014"
), format = "%m-%d-%y")
```

kinMatrix2LongForm *Reformat a kinship matrix into long form*

Description

Part of Group Formation

Usage

```
kinMatrix2LongForm(kinMatrix, removeDups = FALSE)
```

Arguments

kinMatrix	numerical matrix of pairwise kinship values. The row and column names correspond to animal IDs.
removeDups	logical value indication whether or not reverse-order ID pairs be filtered out? (i.e., "ID1 ID2 kin_val" and "ID2 ID1 kin_val" will be collapsed into a single entry if removeDups = TRUE)

Value

A dataframe with columns `id1`, `id2`, and `kinship`. This is the kinship data reformatted from a matrix, to a long-format table.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
reformattedKmat <- kinMatrix2LongForm(kmat, removeDups = FALSE)
nrow(reformattedKmat)
reformattedNoDupsKmat <- kinMatrix2LongForm(kmat, removeDups = TRUE)
nrow(reformattedNoDupsKmat)
```

---

kinship *Generate a kinship matrix*

Description

The function previously had an internal call to the kindepth function in order to provide the parameter pdepth (the generation number). This version requires the generation number to be calculated elsewhere and passed into the function.

Usage

```
kinship(
  id,
  father.id,
  mother.id,
  pdepth,
  sparse = FALSE,
  twinRelations = NULL,
  chrtype = "autosome",
  sex = NULL
)
```

Arguments

id	character vector of IDs for a set of animals.
father.id	character vector or NA for the IDs of the sires for the set of animals.
mother.id	character vector or NA for the IDs of the dams for the set of animals.
pdepth	integer vector indicating the generation number for each animal.
sparse	logical flag. If TRUE, the result is a sparse matrix of the Matrix package with the same values: a ddiMatrix when no two animals have a kinship, otherwise a dgCMatrix. If FALSE, the result is a base matrix.
twinRelations	NULL (default, no-op) or a data.frame with columns id1, id2, code declaring twin pairs (see checkTwinRelations). Only code == "MZ twin" rows affect the computed matrix – "DZ twin"/"UZ twin" rows are accepted but get zero special treatment, matching kinship2's own relation mechanism. A declared MZ-twin pair's coefficient is corrected to equal their shared self-kinship (genetic identity), and the correction is propagated transitively (chained MZ declarations, e.g. A-B and B-C, also correct the undeclared A-C pair) and to every relative computed at a later pedigree depth through either twin – not just the declared pair's own cell. twinRelations is trusted as already validated by checkTwinRelations (this function has no sex parameter of its own and so cannot re-run that validator's full rule set itself); only a cheap self-contained check (both ids present in id) is performed here. See docs/planning/twin-relations-kinship-computation-plan.md for the full design rationale.
chrtype	"autosome" (default) or "x". When "autosome", behavior is unchanged from every prior version of this function – sex is not required and is ignored if supplied. When "x", computes X-chromosome kinship instead: a male's X comes from his mother only (self-kinship 1, since he carries a single copy); a female's X-linked kinship follows the same average-of-parents formula as the autosomal case (self-kinship 0.5, as usual). A founder of unrecognized sex gets NA only on its own diagonal cell and 0 with the other founders, as in kinship2; an animal of unrecognized sex that has parents in the pedigree gets NA kinship with everyone, itself included. The MZ-twin correction (twinRelations) applies inside this branch identically to the autosomal branch. Ported from kinship2's own kinship.default() X-linked branch; see docs/planning/kinship2-supplement-full-reproductio §3 for the full design rationale.
sex	NULL (default) or a character vector, the same length as id, using this package's own internal sex codes (sexCodes): "M"/"F"; any other value, including NA, is treated as unknown). Required when chrtype = "x"; ignored (may be omitted) when chrtype = "autosome".

Details

The rows (cols) of founders are just 0.5 * identity matrix, no further processing is needed for them. Parents must be processed before their children, and then a child's kinship is just the average of the kinship values for his or her parents.

The code for the kinship function was written by Terry Therneau at the Mayo clinic and taken from his website. This function is part of a package written in S (and later ported to R) for calculating kinship and other statistics.

Value

A kinship square matrix

Author(s)

Terry M. Therneau, Mayo Clinic (mayo.edu), original version

All of the code on the original S-Plus kinship function (originally hosted on Terry Therneau's Mayo Clinic software page, offline since at least 2019) was stated to be released under the GNU General Public License (version 2 or later).

The R version became the kinship2 package available on CRAN:

as modified by M Raboin, 2014-09-08 14:44:26

References

https://cran.r-project.org/package=kinship2

The `chrtype = "x"` branch is ported from kinship2's own X-chromosome kinship algorithm, described and worked (Table S2) in its supplementary material:

Sinnwell JP, Therneau TM, Schaid DJ (2014). "The kinship2 R Package for Pedigree Data." *Human Heredity*, 78(2), 91-93.

$Id: kinship.s,v 1.5 2003/01/04 19:07:53 therneau Exp $

Create the kinship matrix, using the algorithm of K Lange, Mathematical and Statistical Methods for Genetic Analysis, Springer, 1997, p 71-72.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
ped
kmat
```

---

kinshipMatricesToKValues

Build a kValue table from a list of kinship matrices

Description

A `kValue` matrix has one row for each pair of individuals in the kinship matrix and one column for each kinship matrix. Thus, in a kinship matrix with 20 individuals the kinship matrix will have 20 rows by 20 columns but only the upper or lower triangle has unique information as the diagonal values are the self-kinship coefficients, $(1 + F)/2$ (0.5 for a non-inbred animal), and the upper triangle has the same values as the lower triangle. The `kValue` table will have 210 rows. The calculation for the number or row in the `kValue` table is $20 + (20 * 19)/2$ rows with the 20 values from the kinship coefficient matrix diagonal and $(20 * 19)/2$ elements from one of either of the two triangles.

Usage

```
kinshipMatricesToKValues(kinshipMatrices)
```

Arguments

`kinshipMatrices`
: list of square matrices of kinship values. May or may not have named rows and columns. The list must contain at least one matrix (an empty list is an error), and every matrix must have its individuals in the same order because the kinship columns are combined by position.

Details

The `kValue` matrix for 1 kinship matrix for 20 individuals will have 210 rows and 3 columns. The first two columns are dedicated to the ID pairs and the third column contains the pair's kinship coefficient.

Thus, the number of rows in the kValues matrix will be $n + n(n-1)/2$ and the number of columns will be 2 plus one additional column for each kinship matrix ($2 + m$, where m is the number of kinship matrices).

Value

A `data.table` with columns `id_1`, `id_2`, and `sim_1` through `sim_m`, one column for each kinship matrix in `kinshipMatrices`. The first two columns contain the IDs of the individuals in the kinship matrices provided to the function and the `sim_` columns contain the corresponding kinship coefficients. Unlike the kinship matrix, each possible pairing of IDs appears once.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_4 <- list(
  id = "J",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
```

```
)
simParent_5 <- list(
  id = "K",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_6 <- list(
  id = "N",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
allSimParents <- list(
  simParent_1, simParent_2, simParent_3,
  simParent_4, simParent_5, simParent_6
)

extractKValue <- function(kValue, id1, id2, simulation) {
  kValue[kValue$id_1 == id1 & kValue$id_2 == id2, paste0(
    "sim_",
    simulation
  )]
}

n <- 10
simKinships <- createSimKinships(ped, allSimParents, pop = ped$id, n = n)
kValue <- kinshipMatricesToKValues(simKinships)
extractKValue(kValue, id1 = "A", id2 = "F", simulation = 1:n)
```

`kinshipMatrixToKValues` *Extract a kValue table from a kinship matrix*

Description

A `kValue` matrix has one row for each pair of individuals in the kinship matrix and one column for each kinship matrix. Thus, in a kinship matrix with 20 individuals the kinship matrix will have 20 rows by 20 columns but only the upper or lower triangle has unique information as the diagonal values are each animal's self-kinship, $(1 + F)/2$ (0.5 when not inbred), and the upper triangle has the same values as the lower triangle. The `kValue` table will have 210 rows. The calculation for the number or row in the `kValue` table is $20 + (20 * 19)/2$ rows with the 20 values from the kinship coefficient matrix diagonal and $(20 * 19)/2$ elements from one of either of the two triangles.

Usage

```
kinshipMatrixToKValues(kinshipMatrix)
```

Arguments

`kinshipMatrix`	square kinship matrix. May or may not have named rows and columns. When the matrix has no row and column names, the IDs are generated (`"A"`, `"B"`, ... `"Z"`, `"A1"`, `"B1"`, ...), not the row and column indices.

Details

The `kValue` matrix for 1 kinship matrix for 20 individuals will have 210 rows and 3 columns. The first two columns are dedicated to the ID pairs and the third column contains the pair's kinship coefficient.

Thus, the number of rows in the kValues matrix will be $n + n(n-1)/2$ and the number of columns will be 3.

Value

A `data.table` with columns `id_1`, `id_2`, and `kinship` where the first two columns contain the IDs of the individuals in the kinship matrix provided to the function and the `kinship` column contains the corresponding kinship coefficient. Unlike the kinship matrix, each possible pairing of IDs appears once.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_4 <- list(
  id = "J",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_5 <- list(
  id = "K",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_6 <- list(
  id = "N",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
allSimParents <- list(
  simParent_1, simParent_2, simParent_3,
  simParent_4, simParent_5, simParent_6
```

```
)

simPed <- makeSimPed(ped, allSimParents)
simKinship <- kinship(
  simPed$id, simPed$sire,
  simPed$dam, simPed$gen
)
kValues <- kinshipMatrixToKValues(simKinship)
```

`lacy1989Ped` *Small hypothetical pedigree (Lacy 1989)*

Description

Small hypothetical pedigree (Lacy 1989)

Usage

```
data(lacy1989Ped)
```

Format

An object of class `data.frame` with 7 rows and 5 columns.

Source

lacy1989Ped is a dataframe containing the small hypothetical pedigree of three founders and four descendants used by Robert C. Lacy in "Analysis of Founder Representation in Pedigrees: Founder Equivalents and Founder Genome Equivalents" Zoo Biology 8:111-123 (1989).

The founders (`A`, `B`, `E`) have unknown parentages and are assumed to have independent ancestries.

id character column of animal IDs

sire the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam the female parent of the animal indicated by the `id` column. Unknown dams are indicated with `NA`

gen generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

population logical vector with all values set TRUE

lacy1989PedAlleles *Gene-drop alleles for lacy1989Ped (5000 iterations)*

Description

A dataframe produced by `geneDrop` on `lacy1989Ped` with 5000 iterations.

Usage

```
data(lacy1989PedAlleles)
```

Format

A dataframe with 14 rows and 5002 columns. There are 5000 columns (`V1` to `V5000`), one for each iteration in `geneDrop` containing alleles randomly selected at each generation of the pedigree using Mendelian rules.

Column 5001 is the `id` column with two rows for each member of the pedigree (2 * 7).

Column 5002 is the `parent` column with values of `sire` and `dam` alternating.

Source

lacy1989Ped is a dataframe containing the small example pedigree used by Robert C. Lacy in "Analysis of Founder Representation in Pedigrees: Founder Equivalents and Founder Genome Equivalents" Zoo Biology 8:111-123 (1989).

loadSiteConfig *Load the site configuration for the modular Shiny application*

Description

Reads the user's site-configuration file (`~/.nprcgenekeepr_config`, or `~/_nprcgenekeepr_config` on Windows) using the tolerant `getSiteInfo` parser, which handles the documented configuration format (comment lines, blank lines, and multi-line / quoted / comma-separated values; see `inst/extdata/examples/example_nprcgenekeepr_config`). The call is wrapped in `tryCatch` so that a missing or malformed configuration file can never crash the application on boot: in that case a warning is logged and `NULL` is returned.

Usage

```
loadSiteConfig()
```

Details

This replaces a former `read.table(sep = "=")` call in the application server that assumed a strict two-column table, could not parse the documented format, and crashed `runGeneKeepR` at startup.

Value

A named list of site information (as returned by `getSiteInfo`) when a configuration file is present and parseable; otherwise `NULL`.

See Also

`getSiteInfo`, `getConfigFileName`, `appServer`

Examples

```
library(nprcgenekeepr)
## Reads ~/.nprcgenekeepr_config (or ~/_nprcgenekeepr_config on
## Windows) if it exists; returns NULL when no config file is
## present, so this is safe to run on any machine.
config <- loadSiteConfig()
config
```

`loadSpeciesOverrides` *Load user-configurable species reproductive-parameter overrides*

Description

Reads the optional species-override settings from the user's site configuration file (`~/.nprcgenekeepr_config`, or `~/_nprcgenekeepr_config` on Windows) and assembles the override tables and fallbacks consumed by the Genetic Value Analysis. The configuration may carry up to three optional keys, each looked up softly (the `getConfigApiKey` pattern – absent keys are not an error and never touch the fixed-schema `getSiteInfo` parser):

- `speciesOverridesPath` – path to a CSV with the four `speciesGestation` columns (`species`, `gestation`, `minMaleBreedingAge`, `minFemaleBreedingAge`; header required, matched by name). A colony lists only the rows (species) it wants to change; the CSV is **merged onto** the bundled table, so every unlisted species keeps its bundled value (not replaced).
- `minBreedingAgeDefault` – numeric fallback (years) for a species absent from the table (bundled built-in 2.0).
- `gestationDefault` – integer fallback (days) for a species absent from the table (bundled built-in 210).

Usage

```
loadSpeciesOverrides()
```

Details

Like `loadSiteConfig`, this never crashes the application on boot: a missing configuration file, a missing override key, or a missing/malformed CSV all fall back to the bundled values (a warning is raised for an unreadable CSV).

Value

A named list with elements `breedingTable`, `gestationTable` (each the merged `speciesGestation`-shaped data.frame, or `NULL` when no CSV is configured), `breedingAgeDefault` (numeric or `NULL`), and `gestationDefault` (integer or `NULL`). A `NULL` element means "use the bundled value / built-in default".

See Also

`loadSiteConfig`, `getSpeciesMinBreedingAge`, `getSpeciesGestation`, `reportGV`

Examples

```
library(nprcgenekeepr)
## Reads optional species overrides from the user's site config
## file; with no config file every element is NULL, meaning "use
## the bundled speciesGestation values / built-in defaults".
overrides <- loadSpeciesOverrides()
str(overrides)
```

---

logModuleEvent *Log module events*

Description

Centralized logging function for Shiny module events. Provides consistent logging format across all modules with configurable log levels.

Usage

```
logModuleEvent(module, message, level = "INFO", ...)
```

Arguments

`module`	character. Name of the module generating the log message.
`message`	character. The log message to record.
`level`	character. Log level: "DEBUG", "INFO", "WARN", or "ERROR". Defaults to "INFO".
`...`	Additional arguments passed to the log message (for sprintf-style formatting).

Details

`"WARN"` and `"ERROR"` entries are always emitted with `message()`. `"INFO"` entries are written to standard output with `cat()` only when `options(nprcgenekeepr.verbose = TRUE)` is set. `"DEBUG"` entries are written the same way only when `options(nprcgenekeepr.debug = TRUE)` is set. Otherwise those two levels produce no output. An unrecognized level is treated as `"INFO"`.

Value

Invisible NULL. Called for side effect of logging.

See Also

`safeExecute` for error-safe execution with logging

Examples

```
logModuleEvent("modInput", "File uploaded successfully")
logModuleEvent("modPedigree", "Processing %d animals", level = "DEBUG", 100)
logModuleEvent("modGeneticValue", "Calculation failed", level = "ERROR")
```

---

makeCEPH *Make a CEPH-style pedigree for each id*

Description

Part of Relations

Usage

```
makeCEPH(id, sire, dam)
```

Arguments

`id`	character vector of the unique identifiers of all individuals; a duplicated id stops the function with a `duplicate row.names` error.
`sire`	character vector, parallel to `id`, with the identifier of each individual's father (`NA` if unknown).
`dam`	character vector, parallel to `id`, with the identifier of each individual's mother (`NA` if unknown).

Details

Creates a CEPH-style pedigree for each id, consisting of three generations: the id, the parents, and the grandparents. Inserts NA for unknown pedigree members. Ids given as numbers or as a factor are taken as their text.

Value

List of lists: fields: id, subfields: parents, pgp, mgp. Pedigree information converted into a CEPH-style list. The top level list elements are the IDs from id. Below each ID is a list of three elements: parents (sire, dam), paternal grandparents (pgp: sire, dam), and maternal grandparents (mgp: sire, dam).

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
pedCEPH <- makeCEPH(ped$id, ped$sire, ped$dam)
head(ped)
head(pedCEPH$F)
```

`makeExamplePedigreeFile` *Write copy of nprcgenekeepr::examplePedigree into a file*

Description

Uses `examplePedigree` data structure to create an example data file. Files of type `"csv"` and `"txt"` are silently overwritten if they already exist. A file of type `"excel"` is never overwritten; if the file already exists, an error is signaled.

Usage

```
makeExamplePedigreeFile(
  file = file.path(tempdir(), paste0("examplePedigree.", c(csv = "csv", txt = "txt",
     excel = "xlsx")[[fileType]])),
  fileType = "csv"
)
```

Arguments

`file`	character vector of length one providing the file name. The default file, in the session's temporary directory, is `examplePedigree.csv`, `examplePedigree.txt` or `examplePedigree.xlsx`, according to `fileType`.
`fileType`	character vector of length one with possible values of `"txt"`, `"csv"`, or `"excel"`. Default value is `"csv"`.

Value

Full path name of file saved.

Examples

```
library(nprcgenekeepr)
pedigreeFile <- makeExamplePedigreeFile()
```

`makeFounderStatsTable` *Create Founder Statistics HTML Table*

Description

Generates an HTML table displaying founder statistics including counts of known founders, female founders, male founders, founder equivalents (FE), and founder genome equivalents (FG). When `founderStats` is `NULL` a paragraph saying that no founder statistics are available is returned instead of a table.

Usage

```
makeFounderStatsTable(founderStats)
```

Arguments

`founderStats`	list containing founder statistics with elements: • `total` - Total number of known founders • `nMaleFounders` - Number of male founders • `nFemaleFounders` - Number of female founders • `fe` - Founder equivalents • `fg` - Founder genome equivalents • `fgSE` - (optional) sampling standard error of `fg`; when finite it is shown inline as `FG +/- SE`

Value

Character string containing HTML table markup.

See Also

`makeGeneticSummaryTable` for genetic value summary

`calcFE` for founder equivalents calculation

`calcFG` for founder genome equivalents

Examples

```
stats <- list(
  total = 50,
  nMaleFounders = 20,
  nFemaleFounders = 30,
  fe = 25.5,
  fg = 22.3
)
html <- makeFounderStatsTable(stats)
```

`makeGeneticDiversityHeatmap` *Make a genetic diversity heat map*

Description

Renders a red/yellow/green stoplight heat map of breeding-group genetic diversity metrics. Each row is a breeding group and each column is a metric; every cell is colored by its color index, where 1 is red (the problem condition), 2 is yellow (watch), and 3 is green (healthy). A metric that is undefined for a group (an `NA` index, such as Production for a group with no breeding-age females) is drawn gray, never as a verdict.

Usage

```
makeGeneticDiversityHeatmap(stats)
```

Arguments

`stats`	A data frame with one row per breeding group. The first column holds the group label, which must be unique (a repeated label stops with an error that names it); every remaining column is a metric whose values are color indices in `c(1, 2, 3)` or `NA`.

Details

This function is agnostic to the number of metric columns: it draws one tile per group-by-metric cell for whatever metric columns it is handed, preserving their input order across the top of the plot. The metric names are written level, centered over their column and in bold, and the group names down the left are the same size (22 pt, 2.5 times the 8.8 pt default), so both stay whole and readable at the size the app draws the plot.

Value

A `ggplot` object: a `geom_tile` heat map with metric headers across the top and group labels down the left, filled red/yellow/green (gray for `NA`) from the color indices.

Examples

```
stats <- data.frame(
  group = c("Group_1", "Group_2"),
  Value = c(1, 3), Origin = c(2, 3),
  Production = c(3, 2), Inbreeding = c(1, 2)
)
p <- makeGeneticDiversityHeatmap(stats)
```

makeGeneticSummaryTable *Create Genetic Summary Statistics HTML Table*

Description

Generates an HTML table displaying summary statistics (Min, Q1, Mean, Median, Q3, Max, Skewness, Kurtosis) for mean kinship and genome uniqueness values.

Usage

```
makeGeneticSummaryTable(geneticValues)
```

Arguments

geneticValues	data.frame containing genetic value columns, in either of two accepted vocabularies: • meanKinship / genomeUniqueness (the legacy names), or • indivMeanKin / gu (reportGV's own $report column names) Both are normalized internally, so reportGV(ped)$report may be passed directly.

Value

Character string containing HTML table markup.

See Also

makeFounderStatsTable for founder statistics

Examples

```
gv <- data.frame(
  meanKinship = c(0.1, 0.2, 0.3, 0.4, 0.5),
  genomeUniqueness = c(0.9, 0.8, 0.7, 0.6, 0.5)
)
html <- makeGeneticSummaryTable(gv)
```

makeGroupMembers *Make the initial groupMembers animal list*

Description

Make the initial groupMembers animal list

Usage

```
makeGroupMembers(numGp, currentGroups, candidates, ped, harem, minAge)
```

Arguments

numGp	integer value indicating the number of groups that should be formed from the list of IDs. There is no default.
currentGroups	list of character vectors of IDs of animals currently assigned to the group. There is no default; use an empty list when no groups are existent.
candidates	character vector of IDs of the animals available for use in the group.
ped	dataframe that is the Pedigree. It contains pedigree information including the IDs listed in ids.
harem	logical variable when set to TRUE, the formed groups have a single male at least minAge old. A group with no male in currentGroups is given a sire chosen at random from candidates; a group that has one keeps him.
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal younger than this age are ignored; an animal exactly this old is not ignored. There is no default.

Value

Initial groupMembers list

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
candidates <- nprcgenekeepr::qcBreeders
## Non-harem: pre-seed group 1 with animals already assigned; a
## second, empty group is initialized ready to be filled.
currentGroups <- list(candidates[1L:3L])
groupMembers <- makeGroupMembers(
  numGp = 2L, currentGroups = currentGroups, candidates = candidates,
  ped = ped, harem = FALSE, minAge = 1L
)
groupMembers
## Harem: each group is seeded with one available male (uses sample()).
set.seed(1L)
haremMembers <- makeGroupMembers(
```

```
  numGp = 2L, currentGroups = list(), candidates = candidates,
  ped = ped, harem = TRUE, minAge = 1L
)
haremMembers
```

---

makeGroupNum *Make the initial grpNum list*

Description

Make the initial grpNum list

Usage

```
makeGroupNum(numGp = 1L)
```

Arguments

numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.

Value

Initial grpNum list

Examples

```
library(nprcgenekeepr)
## Create the initial grpNum list for three groups
grpNum <- makeGroupNum(numGp = 3L)
grpNum
```

---

makeGrpNum *Deprecated alias for makeGroupNum*

Description

makeGrpNum has been renamed to makeGroupNum for consistency with makeGroupMembers. It remains as a deprecated wrapper that issues a warning and then calls makeGroupNum.

Usage

```
makeGrpNum(numGp = 1L)
```

Arguments

numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.

Value

Initial grpNum list

See Also

makeGroupNum

---

makePedigreeDiagramData

Convert a pedigree data frame into visNetwork-ready diagram data

---

Description

Builds the node and edge tables consumed by `visNetwork::visNetwork()` from a pedigree data frame: one node per individual, sex-coded to a node shape, positioned by generation; one directed edge per known sire and one per known dam, pointing from parent to child.

Usage

```
makePedigreeDiagramData(ped, twinRelations = NULL)
```

Arguments

ped	data frame with `id`, `sire`, `dam`, `sex`, and `gen` columns (`sire`/`dam` `NA` for unknown parents; `gen` an integer generation number, 0 for founders, as produced by `findGeneration`). Two optional columns are used when present: a logical `affected` column, which adds an affected-status line to `title` and fills affected nodes with `color.background` `"#CC79A7"` (all other nodes are white, `"#FFFFFF"`); and a character `name` column, which is shown in `label` (long names are truncated) and in `title`.
twinRelations	optional data.frame with columns `id1`, `id2`, `code` (see `checkTwinRelations`). Not validated here; validate with `checkTwinRelations` first. `NULL` (default) adds no connector edges, so `edges` has only `from` and `to`.

Value

A list with two data frames: `nodes` (`id`, `label`, `shape`, `level`, `title`, `color.background`) and `edges` (`from`, `to`, plus `dashes`/`label`/`color` when `twinRelations` is supplied). `title` is an HTML hover-tooltip string giving ID, sex, generation, sire, and dam (and affected status and name when those optional columns are present).

Examples

```
library(nprcgenekeepr)
diagramData <- makePedigreeDiagramData(nprcgenekeepr::examplePedigree)
```

makePedigreeMatingLayout *Combine the mating-unit forest into visNetwork-ready diagram data*

Description

Builds a kinship2-style pedigree layout in which each mating (a sire and dam pair) is drawn as a union node between the parents and their offspring. Returns the same `list(nodes, edges)` shape that `makePedigreeDiagramData` returns, plus a lookup table that maps each duplicate node id to the real individual it stands for. `makePedigreeDiagramData` itself is unaffected; this is a separate function.

Usage

```
makePedigreeMatingLayout(
  ped,
  edgeStyle = c("rectilinear", "direct"),
  twinRelations = NULL,
  kinshipMatrix = NULL
)
```

Arguments

`ped`	data frame with `id`, `sire`, `dam`, `sex`, and `gen` columns, same contract as `makePedigreeDiagramData`. An optional `status` column marks deceased animals: see the `deceased` column of the returned `nodes`. Every column whose name starts `affected` is an affected condition, in the order the data frame holds them, the first four drawn (see `conditions` in the return value). A lone one fills the symbol; two or more split it into sections. (`makePedigreeDiagramData` reads the column `affected` alone.)
`edgeStyle`	one of `"rectilinear"` (default), which routes mate-line and sibship-bar edges through invisible waypoint nodes so they render as a strict right angle, kinship2-style, or `"direct"`, which draws a straight edge from each parent to their mating unit and from each mating unit to each child.
`twinRelations`	optional data.frame with columns `id1`, `id2`, `code` (see `checkTwinRelations`). Not validated here; validate with `checkTwinRelations` first. `NULL` (default) adds no connector edges. A connector always targets the two individuals' real node ids and always renders as a direct edge regardless of `edgeStyle`.
`kinshipMatrix`	optional precomputed kinship matrix (a base `matrix` or a `Matrix`) with row AND column names set to individual ids. When supplied, it replaces this function's internal `kinship()` call as the sole source of the consanguineous-mating-unit flags (`kinshipMatrix[sire, dam] > 0`): `twinRelations` then drives twin connector edges only, so a caller wanting twin-corrected consanguinity must bake it into the matrix (e.g. `kinship(..., twinRelations = ...)`). A sire/dam id absent from the matrix's dimnames leaves that unit at the safe `FALSE` default, exactly as a dangling parent does on the default path. `NULL` (default) computes `kinship(ped$id, ped$sire, ped$dam, ped$gen, twinRelations = twinRelations)`.

Details

Mating-unit nodes render as a small, unlabeled dot with an offspring-count tooltip, visually distinct from the 5 sex-coded shapes. An individual who has to appear more than once in the layout gets duplicate nodes. A duplicate node keeps the real individual's own shape, label and tooltip content (plus a cue that it is a duplicate), so it reads as that individual, and is connected to it by a dashed edge.

In the default `"rectilinear"` edge style, mate-line and sibship-bar edges are routed through invisible waypoint nodes so they render as strict right angles, as kinship2 draws them. In the `"direct"` edge style, edges are straight segments from each parent to the mating unit and from the mating unit to each child.

A simple two-real-parent mating unit (each parent has exactly one mate and both have unambiguous `"M"`/`"F"` sex codes) is drawn with the male parent to the left of the female parent. The final horizontal positions come from a joint quadratic program that preserves each row's provisional left-to-right order, so this rule carries into the rendered layout. It cannot be turned off.

Value

A list with `nodes` (`id`, `label`, `shape`, `title`, `size`, `color.background`, `deceased` (logical, never `NA`: `TRUE` for the node of an animal whose `status` reads `DECEASED`, and for each of its repeat-appearance nodes, `FALSE` for every other node and for all of them when `ped` has no `status` column), `x`, `y`; `color.border` is added under `edgeStyle = "rectilinear"`; `affectedSections` is added when there are two or more conditions: a character string with one character per condition, `"1"` where the animal is affected and `"0"` where it is unaffected or unknown, the same for a repeat-appearance node as for its animal, `""` for a node that is not an animal, never `NA`; with two or more conditions the animals' `color.background` is white, the sections carrying the color), `edges` (`from`, `to`, `dashes`, `smooth.enabled`, `smooth.type`, `smooth.roundness`, `color`, `width`; the `color` and `width` columns are always present once any mating unit exists: a consanguineous mating unit's (`kinship(sire, dam) > 0`) 2 spouse-to-union edges get `"#D55E00"/4`, an Okabe-Ito colorblind-safe vermillion in kinship2's doubled/thickened mate-line convention, and every other edge gets `NA`; `label` is added when `twinRelations` is supplied), and `duplicateToReal` (a named character vector, duplicate node id -> real individual id). Under `edgeStyle = "rectilinear"`, an already-set edge `color` (e.g. the twin connector's or the consanguinity marker's) is preserved when the edge is kept as-is; a marked mate edge that gets replaced by a right-angle projection currently falls back to the generic routing color and width. Also `isolatedIds`, a character vector of ids suppressed from `nodes`/`edges` because they have no known parent, are never a parent, and are not connected by `twinRelations`; it is `character(0)` when nothing was suppressed. When every individual in `ped` is isolated, `nodes`/`edges` are both 0-row and `duplicateToReal` is empty. Also `conditions`, a data frame with one row per drawn condition (`column`, `label` such as "Affected" or "Affected (diabetes)", and its `color`), 0-row when `ped` has no such column, and `conditionsNotDrawn`, the names of any such columns past the fourth.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
layout <- makePedigreeMatingLayout(ped)
```

makeRelationClassesTable *Make a relation classes table from kinship pairs*

Description

Counts the pairs of animals in each relationship class of a long-form kinship table.

Usage

```
makeRelationClassesTable(kin)
```

Arguments

kin	a dataframe with columns id1, id2, kinship, and relation. It is a long-form table of pairwise kinships, with relationship categories included for each pair.

Value

A data.frame with columns Relationship Class and Frequency: the number of pairs in each of the following relationship classes: Parent-Offspring, Full-Siblings, Half-Siblings, Grandparent-Grandchild, Full-Cousins, Cousin - Other, Full-Avuncular, Avuncular - Other, Other, and No Relation. Self pairs are not counted, and classes with no pairs are left out. When no pair of different animals is left to count, the table has its two columns and no rows.

Examples

```
library(nprcgenekeepr)

qcPed <- nprcgenekeepr::qcPed
qcPed <- qcPed[1:50, ] # Comment out for full example
bkmat <- kinship(qcPed$id, qcPed$sire, qcPed$dam, qcPed$gen,
  sparse = FALSE
)
kin <- convertRelationships(bkmat, qcPed)
relClasses <- makeRelationClassesTable(kin)
relClasses$`Relationship Class` <-
  as.character(relClasses$`Relationship Class`)
relClassTbl <- as.data.frame(table(kin$relation[kin$relation != "Self"]))
relClassTbl
```

makeSimPed *Make a simulated pedigree from representative sires and dams*

Description

For each id in allSimParents, an *unknown* (NA) sire or dam is imputed by a random draw from the supplied representative vectors (sires and dams); a *known* sire or dam is preserved unchanged. When a parent is unknown and its representative vector is empty, that parent remains NA.

Usage

```
makeSimPed(ped, allSimParents, verbose = FALSE)
```

Arguments

ped	The pedigree information in data.frame format
allSimParents	list made up of lists where the internal list has the offspring ID id, a vector of representative sires (sires), and a vector of representative dams (dams).
verbose	logical vector of length one that indicates whether or not to print a message when a sire or dam is unknown and there are no representative sires or dams to draw from, so it stays NA.

Details

The algorithm assigns parents randomly from the lists of possible sires and dams and does not prevent a dam from being selected more than once within the same breeding period. While this is probably not introducing a large error, it is not ideal.

Value

the input pedigree as a data.table (all of its columns), with each unknown sire and dam replaced by a random draw where representatives were supplied.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
## For each id below, any unknown sire/dam is replaced by a random
## draw from the supplied representative sires and dams.
allSimParents <- list(
  list(
    id = "A",
    sires = c("s1_1", "s1_2", "s1_3"),
    dams = c("d1_1", "d1_2", "d1_3", "d1_4")
  ),
  list(
    id = "B",
    sires = c("s2_1", "s2_2", "s2_3"),
```

```
    dams = c("d2_1", "d2_2", "d2_3", "d2_4")
  ),
  list(
    id = "E",
    sires = c("s3_1", "s3_2", "s3_3"),
    dams = c("d3_1", "d3_2", "d3_3", "d3_4")
  )
)
set.seed(1)
simPed <- makeSimPed(ped, allSimParents)
simPed
```

`mapIdsToObfuscated` *Map IDs to Obfuscated IDs*

Description

This is not robust: it stops with an error if any ID in `ids` is not one of the names of `map`.

Usage

```
mapIdsToObfuscated(ids, map)
```

Arguments

`ids`	character vector with original IDs
`map`	named character vector where the values are the obfuscated IDs and the vector of names (`names(map)`) is the vector of original names.

Value

A character vector with original IDs replaced by their obfuscated counterparts.

See Also

Other obfuscation: `obfuscateDate()`, `obfuscateGenomicROH()`, `obfuscateGenotypeMatrix()`, `obfuscateId()`, `obfuscateLdBlocks()`, `obfuscateMhcHaplotypes()`, `obfuscatePed()`, `obfuscateTwinRelations()`

Examples

```
set_seed(1)
ped <- qcStudbook(nprcgenekeepr::pedSix)
obfuscated <- obfuscatePed(ped, map = TRUE)
someIds <- c("s1", "s2", "d1", "d1")
mapIdsToObfuscated(someIds, obfuscated$map)
```

`markerExpectedHeterozygosity` *Compute per-locus and population-wide expected heterozygosity*

Description

Computes Nei's gene diversity (the standard $He = 1 - \sum p_i^2$ form, where p_i is the population frequency of allele i at a locus) for each locus from the non-missing genotype calls at that locus, plus the unweighted mean across loci as a population-wide summary. This is the plain (uncorrected) estimator – the small-sample-size-corrected variant (Nei & Roychoudhury 1974) is not computed here.

Usage

```
markerExpectedHeterozygosity(genotypeMatrix)
```

Arguments

`genotypeMatrix`	a character matrix as returned by `buildMarkerGenotypeMatrix`: rows are individual `ids`, columns are loci, and each cell is that individual's two alleles at that locus, sorted and joined by `"/"` (or `NA` if not genotyped at that locus).

Value

A list with two elements: `perLocus`, a named numeric vector of expected heterozygosity per locus (names taken from `colnames(genotypeMatrix)`); and `meanHe`, the unweighted mean of `perLocus` across the loci that have a value. A locus with no genotyped individual gets `NA` and is left out of `meanHe`; `meanHe` is `NA` when no locus has a genotyped individual, the same treatment `markerObservedHeterozygosity` gives an individual with no genotyped locus.

References

Nei, M. (1973). Analysis of gene diversity in subdivided populations. *Proceedings of the National Academy of Sciences USA*, 70(12), 3321-3323. doi:10.1073/pnas.70.12.3321

See Also

`markerObservedHeterozygosity`, `buildMarkerGenotypeMatrix`, `markerKinship`

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("A", "A", "B", "B"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("A", "B", "B", "B"),
  stringsAsFactors = FALSE
```

```
)
genotypeMatrix <- buildMarkerGenotypeMatrix(markerGenotype)
markerExpectedHeterozygosity(genotypeMatrix)
```

`markerFst` *Compute a between-center allele-frequency differentiation statistic (Fst)*

Description

Estimates Hudson's Fst – a population-level, between-center genetic differentiation statistic – from two centers' marker genotype matrices, at each locus genotyped in both centers, pooled across loci. Unlike `resolveCrossCenterIds` (Slice 4), this is a two-*dataset* comparison with no per-animal identity linkage involved – exactly the capability audit's own Dimension 6/8 framing of "identity-by-state / allele-frequency differentiation between centers."

Usage

```
markerFst(genotypeMatrixA, genotypeMatrixB)
```

Arguments

`genotypeMatrixA`
: a character matrix as returned by `buildMarkerGenotypeMatrix` for Center A: rows are individual `id`s, columns are loci, and each cell is that individual's two alleles at that locus, sorted and joined by `"/"` (or `NA` if not genotyped at that locus).

`genotypeMatrixB`
: the same shape as `genotypeMatrixA`, for Center B.

Details

Why Hudson's estimator, not Weir & Cockerham (1984) or Nei's Gst. This slice's Pre-RED research (a 3-angle research pass plus an independent adversarial verification) found:

- Weir & Cockerham (1984) – the estimator most often reached for by name – becomes biased by the *ratio* of the two populations' sample sizes when repurposed for a pairwise comparison between two specific, named populations (as opposed to its original assumption of several populations sharing identical drift): Bhatia, Patterson, Sankararaman & Price (2013) demonstrate this empirically (their Table 1 – resampling one population to a smaller size shifted the Weir-Cockerham estimate by more than 8 standard errors, while Hudson's estimate was essentially unchanged) and explicitly, repeatedly recommend Hudson's estimator for exactly this two-named-population case. This matters concretely here because real per-center marker panels are expected to have modest, plausibly unequal sample sizes (design constraint D2 of the issue #130 plan).

- The adversarial verification pass additionally found that the simple two-term $(MSP-MSG)/(MSP+(n_c-1)MSG)$ form initially considered for "Weir & Cockerham (1984)" is not actually what that name refers to in the literature or in independent software (`scikit-allel`, `hierfstat`) – the true estimator has a third, heterozygosity-driven variance component, and the two-term special case differed from the full estimator by roughly 40% on a hand-computed check. Implementing the truncated formula under a "Weir & Cockerham" citation would have shipped a wrong, mislabeled statistic.
- Nei's Gst is not a single unambiguous formula either: at least three numerically different two-population specializations exist across Nei (1973), Nei & Chesser (1983), and Nei (1986/1987), differing by up to a factor of 2 for the same input data – and neither Bhatia et al. (2013) nor Meirmans & Hedrick (2011) recommend any of them over Hudson's estimator for this scenario.
- Hudson's estimator (as given in closed form by Bhatia et al. 2013) is also the algebraically simplest of the three – a single ratio, no ANOVA/variance-component decomposition – so it carries the least residual formula risk while still matching this package's base-R-only, no-new-hard-dependency design constraint (D2).

Pooling across loci: ratio of sums, not mean of per-locus ratios. Bhatia et al. (2013) show that averaging per-locus Fst values directly is materially biased (their own worked example: a more than 2-fold difference for the same data, driven by loci with small denominators) and explicitly recommend summing numerators and denominators separately before dividing once. `pooledFst` here follows that recommendation; it is deliberately not the mean of `perLocus`.

Reference allele. Each shared, biallelic locus has exactly two alleles by construction (`checkMarkerGenotypeFile` enforces this within each center's own file); the alphabetically-first allele observed across *both* centers' non-missing genotypes at that locus is used as the consistent reference allele for that locus's frequency comparison. (A locus whose combined, cross-center allele set exceeds two distinct alleles – e.g. each center's file is independently biallelic but the two centers used different marker labels for the same locus name – is not defended against; this function assumes the two files genuinely describe the same marker panel.)

Undefined/edge cases. A locus present in only one center's matrix is silently excluded from both `perLocus` and the pooled sums (a plain locus-name intersection, not a warned condition). A locus with zero genotyped individuals in either center, or one that is monomorphic for the same allele in both centers (an exact `0/0`), is excluded with a warning naming the locus – `NA`, not an error, matching `markerKinship`'s and `markerParentageExclusion`'s precedent for the same kind of no-shared-informative-evidence case. A per-locus (or pooled) Fst value may legitimately be negative – sampling noise smaller than expected under the null – and is never clamped to zero.

Value

A list with two elements: `perLocus`, a named numeric vector of Hudson's Fst at each locus genotyped in both centers (names taken from the shared, biallelic loci; `NA` for an excluded locus); and `pooledFst`, the single ratio-of-sums summary across all valid loci (`NA` if no locus is valid).

References

Hudson, R. R., Slatkin, M., & Maddison, W. P. (1992). Estimation of levels of gene flow from DNA sequence data. *Genetics*, 132(2), 583-589. doi:10.1093/genetics/132.2.583

Bhatia, G., Patterson, N., Sankararaman, S., & Price, A. L. (2013). Estimating and interpreting FST: The impact of rare variants. *Genome Research*, 23(9), 1514-1521. doi:10.1101/gr.154831.113

See Also

checkMarkerGenotypeFile, buildMarkerGenotypeMatrix, markerKinship

Examples

```
library(nprcgenekeepr)
genotypeA <- data.frame(
  id = c("A1", "A1", "A2", "A2"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("A", "B", "B", "A"),
  stringsAsFactors = FALSE
)
genotypeB <- data.frame(
  id = c("B1", "B1", "B2", "B2"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("B", "B", "A", "B"),
  allele2 = c("B", "B", "B", "B"),
  stringsAsFactors = FALSE
)
matrixA <- buildMarkerGenotypeMatrix(genotypeA)
matrixB <- buildMarkerGenotypeMatrix(genotypeB)
markerFst(matrixA, matrixB)
```

---

markerKinship *Estimate pairwise kinship directly from marker genotypes (KING-robust)*

Description

Estimates a marker-based kinship matrix, independent of pedigree, using the "between-family" KING-robust estimator of Manichaikul et al. (2010), Equation 11 – the estimator KING, PLINK2, `SNPRelate::snpgdsIBDKING`, and `GENESIS::kingToMatrix` all implement under the name "KING-robust". Unlike kinship (which is purely pedigree-derived), this function never looks at parentage – it is a genotype-only check, matching the issue's own framing of an independent relatedness estimate.

Usage

```
markerKinship(genotypeMatrix)
```

Arguments

genotypeMatrix	a character matrix as returned by buildMarkerGenotypeMatrix: rows are individual ids, columns are loci, and each cell is that individual's two alleles at that locus, sorted and joined by "/" (or NA if not genotyped at that locus).

Details

For a pair of individuals i, j, over the loci genotyped in both:

$$\hat{\phi}_{ij} = 0.5 + \frac{2N_{AaAa} - 4N_{AAaa} - N_{Aa}^{(i)} - N_{Aa}^{(j)}}{4\min(N_{Aa}^{(i)}, N_{Aa}^{(j)})}$$

where N_{AaAa} counts loci at which both individuals are heterozygous, N_{AAaa} counts loci at which they have opposite homozygous genotypes (identity-by-state 0), and $N_{Aa}^{(i)}$/ $N_{Aa}^{(j)}$ count each individual's own heterozygous loci (all restricted to the shared, jointly-non-missing locus set). The estimator requires biallelic markers (see `checkMarkerGenotypeFile`) and is not bounded below by zero – a negative estimate is informative (more divergent ancestry than the reference sample), not an error, and is not clipped.

The diagonal is set to `0.5` by definition (self-kinship), matching `kinship`'s convention, rather than evaluated from the formula above – which divides by zero whenever an individual has no heterozygous loci at all.

When at least one individual in a pair has no heterozygous locus among the loci genotyped in both (so $\min(N_{Aa}^{(i)}, N_{Aa}^{(j)}) = 0$ and the formula's denominator is zero), the pair's kinship is undefined; that pair's entry is `NA` and a warning names the pair.

Value

A numeric `id` x `id` matrix (symmetric, diagonal `0.5`), the same shape `kinship` returns.

References

Manichaikul, A., Mychaleckyj, J. C., Rich, S. S., Daly, K., Sale, M., & Chen, W.-M. (2010). Robust relationship inference in genome-wide association studies. *Bioinformatics*, 26(22), 2867-2873. doi:10.1093/bioinformatics/btq559

See Also

`checkMarkerGenotypeFile`, `buildMarkerGenotypeMatrix`, `kinship`

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("A", "A", "B", "B"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("A", "B", "B", "B"),
  stringsAsFactors = FALSE
)
genotypeMatrix <- buildMarkerGenotypeMatrix(markerGenotype)
markerKinship(genotypeMatrix)
```

markerLdBlock *Compute a descriptive, same-chromosome pairwise LD/block statistic*

Description

Estimates a descriptive linkage-disequilibrium (LD) block statistic (the two columns `Dprime` and `r2`) for every pair of loci on the same chromosome (issue #153, D3b). This is deliberately the *secondary*, exploratory-use statistic in the design's two-metric pair – `markerRealizedRelatednessVariance` is the primary, genuinely pedigree-valid metric. No CRAN package is both pedigree-aware and multiallelic-capable (issue #153 design doc sec 2.12), so classical LD theory's random-mating assumption is a genuine, documented compromise for a pedigreed colony sample (D8) – the `caveat` column on every returned row is not decorative and must not be dropped.

Usage

```
markerLdBlock(genotypeMatrix, locusMetadata, founderIds = NULL)
```

Arguments

`genotypeMatrix`	a character matrix as returned by `buildMarkerGenotypeMatrix`: rows are individual `id`s, columns are loci, and each cell is that individual's two alleles at that locus, sorted and joined by `"/"` (or `NA` if not genotyped at that locus).
`locusMetadata`	dataframe as returned by `checkLocusMetadata`: at least `locus` and `chrom` columns. Loci with `NA` `chrom` are excluded from pairing.
`founderIds`	optional character vector of ids to which the computation should be restricted (e.g. `getFounders(ped)`); `NULL` (default) uses every individual in `genotypeMatrix`.

Details

Because the genotype matrix is unphased, `markerLdBlock` estimates two-locus phase frequencies for each pair via a multiallelic maximum-likelihood (EM) estimator, generalizing the classic biallelic two-locus EM (Excoffier & Slatkin 1995) to arbitrary allele counts per locus. Per-allele-pair D and its standardized D-prime use the classic Lewontin (1964) standardization; the per-pair `Dprime` is Hedrick's (1987) frequency-weighted average of the absolute standardized per-allele-pair values across all allele pairs, and `r2` is a chi-squared/Cramer's-phi-squared-style multiallelic generalization that reduces exactly to the classic biallelic r^2 when both loci happen to be biallelic.

Only same-chromosome locus pairs are computed – `locusMetadata`'s `pos`/`cM` columns are not used (D2's own finding that locus-order metadata is typically sparse or absent for real colony panels). A pair with fewer than 2 individuals genotyped at both loci (after any `founderIds` restriction) returns `NA` with a named warning, matching `markerFst`'s precedent for an insufficient-evidence pair – not a `stop()`. The function does `stop()` when no locus has a non-`NA` `chrom`. A locus missing from either `locusMetadata` or `genotypeMatrix` is dropped silently, and when no chromosome has two or more loci the result is a zero-row dataframe with the full column set.

Value

A dataframe with one row per same-chromosome locus pair: `locus1`, `locus2`, `chrom`, `Dprime`, `r2`, `nUsed` (individuals genotyped at both loci), `idsUsed` (comma-joined ids, populated only when `founderIds` is supplied – `NA` otherwise), and `caveat` (a fixed, non-droppable descriptive-statistic warning on every row).

References

Excoffier, L., & Slatkin, M. (1995). Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Molecular Biology and Evolution*, 12(5), 921-927. doi:10.1093/oxfordjournals.molbev.a040269

Hedrick, P. W. (1987). Gametic disequilibrium measures: proceed with caution. *Genetics*, 117(2), 331-341.

Weir, B. S. (1996). *Genetic Data Analysis II: Methods for Discrete Population Genetic Data*. Sinauer Associates.

See Also

`buildMarkerGenotypeMatrix`, `checkLocusMetadata`, `markerRealizedRelatednessVariance`, `obfuscateLdBlocks`

Examples

```
library(nprcgenekeepr)
genotype <- checkLinkageMarkerGenotypeFile(data.frame(
  id = rep(c("W", "X", "Y", "Z"), 2),
  locus = rep(c("L1", "L2"), each = 4),
  allele1 = c("A", "A", "A", "A", "M", "M", "N", "N"),
  allele2 = c("A", "B", "A", "B", "M", "N", "M", "N"),
  stringsAsFactors = FALSE
))
genotypeMatrix <- buildMarkerGenotypeMatrix(genotype)
locusMetadata <- checkLocusMetadata(data.frame(
  locus = c("L1", "L2"), chrom = c("1", "1"), pos = c(NA, NA),
  stringsAsFactors = FALSE
))
markerLdBlock(genotypeMatrix, locusMetadata)
```

---

markerObservedHeterozygosity

Compute per-animal observed heterozygosity from marker genotypes

---

Description

Computes, for each individual, the fraction of its genotyped (non-missing) loci at which it is heterozygous – the standard "individual/multilocus heterozygosity" statistic conventionally reported per animal (as opposed to the per-locus population framing, which this function does not compute; see `markerExpectedHeterozygosity` for the population-level counterpart). A raw empirical proportion, not an estimate of a hidden population parameter, so no bias correction applies.

Usage

```
markerObservedHeterozygosity(genotypeMatrix)
```

Arguments

genotypeMatrix	a character matrix as returned by buildMarkerGenotypeMatrix: rows are individual ids, columns are loci, and each cell is that individual's two alleles at that locus, sorted and joined by "/" (or NA if not genotyped at that locus).

Value

A named numeric vector, one observed-heterozygosity value per id (names taken from rownames(genotypeMatrix)), each in [0, 1]. An individual with no non-missing loci returns NA.

References

Nei, M. (1973). Analysis of gene diversity in subdivided populations. *Proceedings of the National Academy of Sciences USA*, 70(12), 3321-3323. doi:10.1073/pnas.70.12.3321

See Also

markerExpectedHeterozygosity, buildMarkerGenotypeMatrix, markerKinship

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("A", "A", "B", "B"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("A", "B", "B", "B"),
  stringsAsFactors = FALSE
)
genotypeMatrix <- buildMarkerGenotypeMatrix(markerGenotype)
markerObservedHeterozygosity(genotypeMatrix)
```

---

markerParentageExclusion

Flag Mendelian-inconsistent recorded parents from marker genotypes

---

Description

For each animal in pedigree with a recorded dam and/or sire that is also genotyped, compares the animal's marker genotype to that recorded parent's, locus by locus, and counts loci at which no shared allele is possible under simple Mendelian inheritance ("opposite homozygotes" – the animal and the candidate parent are each homozygous for a *different* allele). Aggregates to a per-pair exclusion count and a flagged decision, directly targeting the issue's named ~5% dam-misidentification problem: a recorded parent whose genotype evidence contradicts the pedigree.

Usage

```
markerParentageExclusion(genotypeMatrix, pedigree, maxExclusions = 2L)
```

Arguments

genotypeMatrix	a character matrix as returned by buildMarkerGenotypeMatrix: rows are individual ids, columns are loci, and each cell is that individual's two alleles at that locus, sorted and joined by "/" (or NA if not genotyped at that locus).
pedigree	a data frame with (at least) columns id, sire, and dam – the standard pedigree shape used throughout this package. sire/dam may be NA for an unrecorded parent.
maxExclusions	integer; the maximum number of Mendelian- inconsistent loci tolerated before a recorded parent is flagged as excluded. Default 2L (flag only at 3 or more).

Details

A locus contributes to the exclusion count only when both the animal and the candidate parent are genotyped there *and* both are homozygous for different alleles – the same "informative conflict" definition verified against the ICAR/ISAG cattle-SNP parentage-verification standard at this function's Pre-RED (a heterozygous genotype at either individual is never, by itself, Mendelian-inconsistent with a biallelic parent genotype). Loci where either individual is not genotyped are excluded from both the numerator and the denominator.

maxExclusions is the maximum number of Mendelian-inconsistent loci *tolerated* before a recorded parent is flagged as excluded (i.e. flagging requires exclusionCount > maxExclusions) – a single mismatching locus is not, by itself, evidence the recorded parent is wrong, since ordinary genotyping error or mutation can produce an isolated conflict even for a true parent-offspring pair. The default of 2 (flag only at 3 or more inconsistent loci) is grounded in Cifuentes et al. (2006) and the real captive-macaque-colony parentage precedent of de Groot et al. (2025), both cited below – it is a raw locus count calibrated to small/moderate marker panels and typical genotyping error rates reported in that literature; it does not scale with the number of loci actually typed or with this package's (currently unmeasured) per-locus genotyping-error rate, so panels much larger or noisier than that should retune maxExclusions rather than rely on the shipped default.

When an animal and its recorded parent share zero jointly-genotyped loci, the exclusion count is undefined; that pair's exclusionCount and flagged are NA and a warning names the pair (mirroring markerKinship's precedent for the same kind of no-shared-evidence case). A recorded parent that is NA (unknown) or that has no row in genotypeMatrix (never genotyped) is silently skipped – no row is emitted for that pair, since there is no genotype evidence to check.

Value

A data frame, one row per (offspring, recorded-parent) pair for which both individuals are genotyped, with columns id (the offspring), parentId, role ("dam" or "sire"), exclusionCount, nLoci (the number of jointly-genotyped loci the count is based on), and flagged (the canonical boolean vocabulary, matching reportGV()'s flagged column). A pair with an unrecorded or ungenotyped parent has no row at all. A data frame with zero rows (but the full column shape) is returned when no pair is checkable.

References

Cifuentes, L. O., Martinez, E. H., Acuna, M. P., & Jonquera, H. G. (2006). Probability of exclusion in paternity testing: time to reassess. *Journal of Forensic Sciences*, 51(2), 349-350. doi:10.1111/j.15564029.2006.00046.x

de Groot, N. G., de Vos-Rouweler, A. J. M., Heijmans, C. M. C., et al. (2025). Genetic Conservation and Population Management of Non-Human Primates: Parentage Determination Using Seven Microsatellite-Based Multiplexes. *Ecology and Evolution*, 15(4), e71216. doi:10.1002/ece3.71216

See Also

checkMarkerGenotypeFile, buildMarkerGenotypeMatrix, markerKinship

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("A", "A", "B", "B"),
  locus = c("L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A"),
  allele2 = c("A", "B", "B", "B"),
  stringsAsFactors = FALSE
)
genotypeMatrix <- buildMarkerGenotypeMatrix(markerGenotype)
pedigree <- data.frame(id = "B", sire = NA_character_, dam = "A",
                       stringsAsFactors = FALSE)
markerParentageExclusion(genotypeMatrix, pedigree)
```

---

markerParentageLikelihood

Rank candidate replacement parents by a multilocus likelihood (LOD) score

---

Description

For each recorded parent markerParentageExclusion flags as Mendelian-inconsistent with its offspring's marker genotype, ranks candidate replacement parents using a CERVUS-style multilocus likelihood-ratio (LOD) score – the field's own answer to exactly this problem shape (Meagher & Thompson 1986; operationalized by Marshall, Slate, Kruuk & Pemberton 1998), independently validated as the captive -primate-colony domain's de facto standard by de Groot et al. (2025), a real captive macaque colony precedent already cited by markerParentageExclusion. This is a report-only diagnostic: it never writes to pedigree$sire/pedigree$dam – "requires curator review rather than silently rewriting a pedigree," per the issue's own words. markerParentageExclusion() itself is untouched and remains the independent Mendelian-exclusion check.

Usage

```
markerParentageLikelihood(
  genotypeMatrix,
  pedigree,
  id = NULL,
  role = NULL,
  candidates = NULL,
  minLoci = 4L,
  maxExclusions = 2L
)
```

Arguments

genotypeMatrix	a character matrix as returned by buildMarkerGenotypeMatrix: rows are individual ids, columns are loci, and each cell is that individual's two alleles at that locus, sorted and joined by "/" (or NA if not genotyped at that locus).
pedigree	a data frame with (at least) columns id, sire, and dam – the standard pedigree shape used throughout this package.
id	optional single offspring id to score. When NULL (together with role, the default), every (offspring, role) pair markerParentageExclusion flags is auto-detected and scored.
role	optional, one of "sire"/"dam" – the recorded -parent slot to score candidates for. Required together with id when scoring a single slot on demand (e.g. a curator's proactive check on an animal that is not (yet) flagged).
candidates	optional character vector of candidate ids, used only together with an explicit id/role. When NULL (the default), candidates come from getPotentialParents's own sires/dams list for that id (auto-detection always uses this default – candidates has no effect in auto-detect mode).
minLoci	integer; candidates scored on fewer than this many jointly -genotyped loci are flagged lowPower = TRUE. Default 4L.
maxExclusions	integer; passed through to markerParentageExclusion for auto-detection, and used identically here for each candidate's own excluded diagnostic and for deciding whether a genotyped other-parent qualifies for trio conditioning. Default 2L.

Details

Formula. For each jointly-genotyped locus, the per-locus likelihood ratio compares H1 (the candidate is the true parent) against H2 (the candidate is an unrelated individual, drawn at random from the colony-wide reference population) using ordinary Mendelian transmission probabilities and Hardy-Weinberg population genotype frequencies (well -defined because checkMarkerGenotypeFile guarantees exactly two alleles per locus). When the offspring's other recorded parent is genotyped and not itself Mendelian-excluded (its own opposite-homozygote count against the offspring does not exceed maxExclusions), it is incorporated as a known second parent at that locus (a trio likelihood, matching Marshall et al. 1998's own formula, which conditions on a known mother when scoring candidate fathers); otherwise that locus falls back to a candidate-only (dyad) likelihood using the population allele frequency in place of the unknown second parent. LOD is the sum of

per-locus log-likelihood ratios across all jointly-genotyped loci between the offspring and the candidate.

Reference-population allele frequencies are colony-wide, computed from every non-missing genotype call at that locus in `genotypeMatrix` (the same "whatever subset is passed in" convention `markerExpectedHeterozygosity` already uses) – matching the direct captive-colony precedent of de Groot et al. (2025), who used colony-wide frequencies rather than an external reference population. Structured/inbred captive pedigrees can violate the underlying Hardy-Weinberg assumption, biasing the H2 null; this is a documented limitation, not corrected here.

No simulation-calibrated percentage confidence is reported. CERVUS -style confidence statistics require ~100,000-simulation Monte Carlo calibration per parameter set – a materially larger, separable engineering investment. Instead, `delta` (the LOD gap to the next -ranked candidate within the same offspring/role group) and `nLociUsed` are reported alongside the raw `LOD`, explicitly uncalibrated. **A single Mendelian-incompatible (opposite-homozygote) locus drives** `LOD` **to exactly** `-Inf` – a true probability-zero Mendelian impossibility under this no-genotyping-error formula – which can happen even when `excluded` is `FALSE` (i.e. the candidate's own opposite-homozygote count is at or below `maxExclusions`, the tolerance threshold `markerParentageExclusion` itself uses for ordinary genotyping error): `LOD` has no equivalent error tolerance in this formula. A genotyping-error-tolerant extension (Kalinowski, Taper & Marshall 2007) is deliberately deferred, pending independent re-verification of that paper's own eqns 1-2 against its primary source (this session's Pre-RED found a spotted internal inconsistency in that paper's own Appendix, and its 2010 corrigendum was unretrievable).

Small marker panels (2-10 loci) are underpowered. At this package's realistic panel sizes, LOD-based assignment sits inside the literature's own documented underpowered zone, and a full/half sibling of the true parent can plausibly outrank it. `minLoci` is a fixed, literature-informed, user-overridable heuristic (mirroring `markerParentageExclusion`'s own `maxExclusions` precedent, not a mathematically-derived cutoff): candidates scored on fewer than `minLoci` jointly-genotyped loci are flagged `lowPower = TRUE` rather than silently ranked as if fully powered.

Ties at small panel sizes are surfaced explicitly – every tied candidate appears as its own row, never silently deduplicated or dropped. Two candidates exactly tied (including two `-Inf` values, a real, hand-verified case) report `delta = 0` between them, not `NaN`.

A candidate/offspring pair sharing zero genotyped loci is reported, not silently dropped: `nLociUsed = 0L`, `LOD = NA_real_`, `lowPower = TRUE`, `excluded = NA` (undefined, mirroring `markerParentageExclusion`'s own convention for the same zero-shared-evidence case), with a warning naming the pair.

Value

A data frame, one row per (offspring `id`, `role`, `candidateId`), ranked by `LOD` descending within each (`id`, `role`) group: `id`, `role`, `candidateId`, `LOD`, `delta` (gap to the next-ranked candidate in the group; `NA` for the lowest-ranked row), `nLociUsed`, `excluded` (the same opposite-homozygote diagnostic `markerParentageExclusion` uses, parameterized against this candidate – never overloading that function's own `flagged` column), `lowPower`. A zero-row data frame (full column shape) is returned when no pair is checkable. This function never modifies `pedigree`.

References

Meagher, T. R., & Thompson, E. (1986). The relationship between single parent and parent pair genetic likelihoods in genealogy reconstruction. *Theoretical Population Biology*, 29(1), 87-106. doi:10.1016/00405809(86)900067

Marshall, T. C., Slate, J., Kruuk, L. E. B., & Pemberton, J. M. (1998). Statistical confidence for likelihood-based paternity inference in natural populations. *Molecular Ecology*, 7(5), 639-655. doi:10.1046/j.1365294x.1998.00374.x

Kalinowski, S. T., Taper, M. L., & Marshall, T. C. (2007). Revising how the computer program CERVUS accommodates genotyping error increases success in paternity assignment. *Molecular Ecology*, 16(5), 1099-1106. doi:10.1111/j.1365294X.2007.03089.x

See Also

markerParentageExclusion, getPotentialParents, buildMarkerGenotypeMatrix

Examples

```
library(nprcgenekeepr)
markerGenotype <- data.frame(
  id = c("O", "O", "D", "D", "C1", "C1", "C2", "C2"),
  locus = c("L1", "L2", "L1", "L2", "L1", "L2", "L1", "L2"),
  allele1 = c("A", "A", "A", "A", "A", "A", "B", "B"),
  allele2 = c("A", "B", "B", "A", "A", "B", "B", "B"),
  stringsAsFactors = FALSE
)
genotypeMatrix <- buildMarkerGenotypeMatrix(markerGenotype)
pedigree <- data.frame(id = "O", sire = "C2", dam = "D",
                       stringsAsFactors = FALSE)
markerParentageLikelihood(genotypeMatrix, pedigree, id = "O", role = "sire",
                          candidates = c("C1", "C2"), minLoci = 1L)
```

---

markerRealizedRelatednessVariance

Estimate the variance of realized relatedness around pedigree kinship

---

Description

Pedigree-expected relatedness (this package's existing kinship output, doubled) is only an average – the *actual* proportion of genome shared identical-by-descent (IBD) between two relatives varies around that expectation because of Mendelian sampling and linkage (finite chromosome number/map length creates covariance in IBD status among nearby loci). This function estimates that variance for Parent-Offspring, Full-Siblings, and Half-Siblings pairs (issue #153, D3a) – the closed-form solution of Hill & Weir (2011), extending the pedigree kinship this package already computes rather than requiring a new population-genetics framework. It lists only those three kinds of pairs; every other pair (grandparent, cousin, avuncular, more distant, unrelated, self) is left out, and convertRelationships lists every pair.

Usage

```
markerRealizedRelatednessVariance(kmat, ped, nChr, mapLength, ids = NULL)
```

Arguments

kmat	square kinship matrix, as produced by kinship.
ped	dataframe with (at least) id, sire, and dam columns, as used by convertRelationships. Ids given as numbers or as a factor, here or in ids, are taken as their text.
nChr	chromosome count (e.g. autosome count for the species), meant to be a whole number. Only a single positive numeric value is enforced, so a non-integer such as 20.5 is accepted.
mapLength	numeric; total autosomal genetic map length in Morgans. Must be a single positive value.
ids	character vector of IDs or NULL to which the analysis should be restricted, as in convertRelationships.

Details

Pairs are named by the rules of convertRelationships and listed in its order, but only an animal with each of its parents and two animals sharing a parent are looked at, so the table of every pair is never built. The variance combines nChr chromosomes, each approximated as the average length mapLength / nChr – Hill & Weir (2011) give an exact weighted-sum combination rule (their equation 5) only for lineal descendants; for Full-Sib/Half-Sib pairs they state in prose that this equal-length approximation closely matches a real heterogeneous-length genome, which this package's own PRE-RED research verified numerically against their published human-genome Table 2 (within ~2%; see the test file header for the full derivation).

Value

A dataframe with one row for each "Parent-Offspring", "Full-Siblings" or "Half-Siblings" pair and columns id1, id2, kinship, relation, R (pedigree relationship, 2 * kinship), varR (the realized-relatedness variance estimate), and sdR (its square root).

References

Hill WG, Weir BS. 2011. Variation in actual relationship as a consequence of Mendelian sampling and linkage. Genetics Research 93(1):47-64.

See Also

kinship, convertRelationships

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
## Rhesus macaque autosome count/map length are used only as an example
## scale -- callers should supply values appropriate to their own species.
markerRealizedRelatednessVariance(kmat, ped, nChr = 20L, mapLength = 28)
```

`meanKinship` *Calculate mean kinship for each animal in a kinship matrix*

Description

Part of Genetic Value Analysis

Usage

```
meanKinship(kmat)
```

Arguments

`kmat`	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.

Details

The mean kinship of animal *i* is

$$MK_i = \Sigma f_i j / N$$

, in which the summation is over all animals, *j*, including the kinship of animal *i* to itself. `NA` values are dropped, so *N* is the number of non-`NA` kinship values in the animal's column, not the number of animals.

Value

A named numeric vector of average kinship coefficients for each animal ID. Elements are named with the IDs from the columns of kmat.

References

Ballou JD, Lacy RC. 1995. Identifying genetically important individuals for management of genetic variation in pedigreed populations, p 77-111. In: Ballou JD, Gilpin M, Foose TJ, editors. Population management for survival and recovery. New York (NY): Columbia University Press.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
head(meanKinship(kmat))
```

`mhcHaplotypeCarriers` *List the animals carrying each MHC haplotype*

Description

Builds the carrier detail table for a wide per-animal MHC haplotype designation file (the `checkMhcHaplotypeFile` format): one row per (haplotype x carrying animal), so a colony manager can go from "which haplotypes are rare" (`mhcHaplotypeFrequency`'s summary) to "which animals do I manage." A homozygous animal appears once per haplotype. The haplotype universe is the summary's – the distinct *certain* haplotypes – so a label observed only as uncertain calls has no carrier rows, matching the summary's own exclude-and-disclose rule.

Usage

```
mhcHaplotypeCarriers(
  genotype,
  rareOnly = TRUE,
  rareFrequencyThreshold = 0.01,
  rareCarrierThreshold = 2L
)
```

Arguments

`genotype`	dataframe with wide-format MHC haplotype data as validated by `checkMhcHaplotypeFile` (which this function re-runs defensively): columns `id`, `haplotype1`, `haplotype2`, one row per animal.
`rareOnly`	logical; when `TRUE` (the default) only the haplotypes `mhcHaplotypeFrequency` flags rare at the given thresholds are listed, when `FALSE` every summary haplotype is.
`rareFrequencyThreshold`	single non-negative number, passed to `mhcHaplotypeFrequency`. Default `0.01`.
`rareCarrierThreshold`	single non-negative number, passed to `mhcHaplotypeFrequency`. Default `2L`.

Details

The `uncertain` column is a per-(haplotype x animal) disclosure: `FALSE` when the animal has at least one certain call of that haplotype, `TRUE` when its carriage is only provisional (every call it has of that haplotype carries the trailing-? uncertain marker). Provisional carriers are listed here – for a rare haplotype, a provisionally typed carrier is exactly what a manager wants to see – but they are never counted in `mhcHaplotypeFrequency`'s `nCarriers` or its rare-flag arithmetic. Retaining animals that carry rare variants is a distinct management objective from maintaining heterozygosity (Allendorf 1986; Lacy, Ballou & Pollak 2012), which is what a per-animal carrier list is for.

Value

A dataframe with columns `haplotype` (character), `id` (character), and `uncertain` (logical), one row per (haplotype x carrying animal), ordered by haplotype then id.

References

Allendorf, F. W. (1986). Genetic drift and the loss of alleles versus heterozygosity. *Zoo Biology*, 5(2), 181-190. doi:10.1002/zoo.1430050212

Lacy, R. C., Ballou, J. D., & Pollak, J. P. (2012). PMx: software package for demographic and genetic analysis and management of pedigreed populations. *Methods in Ecology and Evolution*, 3(2), 433-437. doi:10.1111/j.2041210X.2011.00148.x

See Also

mhcHaplotypeFrequency, checkMhcHaplotypeFile, rhesusGenotypes

Examples

```
library(nprcgenekeepr)
## Carriers of the rare haplotypes, at the default thresholds
rareCarriers <- mhcHaplotypeCarriers(rhesusGenotypes)
head(rareCarriers)
## Every haplotype's carriers
allCarriers <- mhcHaplotypeCarriers(rhesusGenotypes, rareOnly = FALSE)
```

---

mhcHaplotypeFrequency *Summarize MHC haplotype frequencies and flag rare haplotypes*

Description

Computes, from a wide per-animal MHC haplotype designation table (the checkMhcHaplotypeFile format), a per-haplotype summary – copy count, carrier count, uncertain-call count, frequency, and a rare flag – plus the file-level counts that make the frequencies interpretable. Frequencies follow the HLA/NHP-MHC convention of a chromosomes-among-genotyped denominator (Solberg et al. 2008; Doxiadis et al. 2013): the number of *certain* calls, i.e. 2 x animals minus missing minus uncertain calls. Uncertain calls (a trailing ? on a designation) are excluded from copy counts, carrier counts, and frequencies but always disclosed – per haplotype in `nUncertain` and file-wide in `counts` – and a haplotype observed *only* as uncertain calls gets no summary row (it is never counted as a distinct haplotype; it remains visible in the file-level counts).

Usage

```
mhcHaplotypeFrequency(
  genotype,
  rareFrequencyThreshold = 0.01,
  rareCarrierThreshold = 2L
)
```

Arguments

genotype	dataframe with wide-format MHC haplotype data as validated by checkMhcHaplotypeFile (which this function re-runs defensively): columns id, haplotype1, haplotype2, one row per animal.
rareFrequencyThreshold	single non-negative number; a haplotype with frequency at or below it is flagged rare. Default 0.01.
rareCarrierThreshold	single non-negative number; a haplotype with that many or fewer carriers is flagged rare. Default 2L.

Details

A haplotype is flagged rare when its frequency is at or below rareFrequencyThreshold OR its carrier count is at or below rareCarrierThreshold – a dual criterion with direct precedent in the combined frequency and observation-count bins of the CIWD 3.0 catalog (Hurley et al. 2020). Each leg is anchored: about 0.01 is the published nonhuman-primate MHC "rare" usage (Kanthaswamy et al. 2026; Doxiadis et al. 2013), and a haplotype carried by two or fewer animals can be lost to two removals regardless of frequency, the allele-retention framing of conservation management. At small colony scale the carrier leg does the flagging (no observed haplotype can have frequency at or below 0.01 when fewer than about 100 chromosomes are genotyped), while at registry scale the frequency leg takes over. Both defaults are working-filter settings, not scientific claims.

A homozygous animal contributes two copies and one carrier. Carrier counts include only animals with at least one *certain* call of the haplotype – an animal whose only call of a haplotype is uncertain is a provisional carrier, listed (and flagged) by mhcHaplotypeCarriers but never counted here. Haplotype designations are opaque labels throughout: never split, parsed, or matched against MHC region names (see checkMhcHaplotypeFile).

Value

A list with two elements. summary: a dataframe with one row per distinct certain haplotype, ordered by haplotype label – haplotype (character), nCopies (integer, certain calls), nCarriers (integer, animals with at least one certain call), nUncertain (integer, uncertain calls of this haplotype), frequency (nCopies / denominator), and isRare (logical, the dual criterion above). counts: a one-row dataframe of file-level totals – nAnimals, nCalls, nMissing, nUncertain, denominator (all integer).

References

Kanthaswamy, S., et al. (2026). Next-generation short-read sequencing reveals impacts on major histocompatibility complex diversity resulting from differences in captive rhesus macaque (*Macaca mulatta*) colony expansion strategies. *American Journal of Primatology*, 88(1), e70108. doi:10.1002/ajp.70108

Hurley, C. K., et al. (2020). Common, intermediate and well-documented HLA alleles in world populations: CIWD version 3.0.0. *HLA*, 95(6), 516-531. doi:10.1111/tan.13811

Solberg, O. D., Mack, S. J., Lancaster, A. K., Single, R. M., Tsai, Y., Sanchez-Mazas, A., & Thomson, G. (2008). Balancing selection and heterogeneity across the classical human leukocyte

antigen loci: A meta-analytic review of 497 population studies. *Human Immunology*, 69(7), 443-464. doi:10.1016/j.humimm.2008.05.001

Doxiadis, G. G., et al. (2013). Haplotype diversity generated by ancient recombination-like events in the MHC of Indian rhesus macaques. *Immunogenetics*, 65(8), 569-584. doi:10.1007/s00251013-07078

See Also

mhcHaplotypeCarriers, checkMhcHaplotypeFile, rhesusGenotypes

Examples

```
library(nprcgenekeepr)
result <- mhcHaplotypeFrequency(rhesusGenotypes)
head(result$summary)
result$counts
```

---

modBreedingGroupsServer

Breeding Groups Module - Server Function

Description

Server logic for breeding group formation using the groupAddAssign algorithm. This module integrates with the kinship-based maximal independent set (MIS) algorithm to form optimal breeding groups that minimize relatedness within groups while maximizing group sizes.

Usage

```
modBreedingGroupsServer(
  id,
  pedigree,
  geneticValues = NULL,
  kinshipMatrix = NULL,
  kinshipOverrides = NULL,
  twinRelations = NULL,
  kinshipFor = NULL
)
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigree	reactive returning pedigree data frame with columns: id, sire, dam, sex, and optionally birth, exit, gen.

`geneticValues` optional reactive returning genetic value results from `modGeneticValueServer`. Used to source the `topRanked` animal-source candidate list and, with any animal source, for the "Genetic-value floor" inclusion criterion, which drops "Low Value" animals and IDs absent from the report. Group formation halts until it is available in either of those cases. Unrelated to kinship.

`kinshipMatrix` optional reactive returning a kinship matrix, used unchanged, so it should already carry any twin correction and kinship overrides. It is used before `kinshipFor`; with neither, the module calculates kinship from the pedigree.

`kinshipOverrides` optional reactive returning a validated outside-information kinship-override data frame (`id1`, `id2`, `kinship`); see `applyKinshipOverrides`. When the module recomputes kinship from the pedigree (neither `kinshipMatrix` nor `kinshipFor` gives a matrix), the overrides are applied to that matrix so group formation reflects them regardless of tab order. `NULL` (the default) is a no-op. A provided `kinshipMatrix` is expected to already carry overrides applied at its source.

`twinRelations` optional reactive returning a validated twin/zygosity sidecar data.frame (`id1`, `id2`, `code`); see `checkTwinRelations`. When the module recomputes kinship from the pedigree (neither `kinshipMatrix` nor `kinshipFor` gives a matrix), it is passed straight through to `kinship` so group formation reflects a declared MZ-twin pair's corrected identity regardless of tab order (BL-N Slice 3). `NULL` (the default) is a no-op. A provided `kinshipMatrix` is expected to already reflect it at its source.

`kinshipFor` optional function of a character vector of animal IDs that returns those animals' kinship matrix (row and column names are the IDs). At Form Groups the module asks it for the candidates plus any seed animals, and keeps only that matrix with the results. `appServer` passes one that computes kinship from those animals and their ancestors only, with the twins and overrides applied. A provided `kinshipMatrix` is used first; with neither, the module calculates kinship from the pedigree. `NULL` (the default) is not used.

Details

The module supports multiple configuration options:

- **Animal source**: "Top ranked", "All available" or "My own list" (animal IDs typed or pasted by the user)
- **Living animals only**: Every source draws only on animals that are alive: those with a `status` of `ALIVE` when the pedigree has a `status` column, otherwise those with no `exit` date, and every animal when it has neither column. The cut to the top N animals or the genetic-value floor comes after this, so "Top ranked" takes the best-ranked living animals. When no animal in the source is alive the tab shows an error notice and forms no groups. Animals typed into a seed group are not drawn from this pool and are accepted as typed
- **Inclusion criterion**: Include animals by "Top N ranked" (with the number of top animals) or "Genetic-value floor"
- **Group counts and ages**: The number of groups and the minimum breeding age
- **Simulations and exhaustive mode**: The number of simulations; "Exhaustive enumeration mode" is offered only when the number of groups is 1 and the sex ratio is "none"

- **Seed groups**: Optionally seed groups with specific animals
- **Kinship threshold**: Maximum allowed kinship within groups
- **Harem mode**: Form groups with exactly one male each
- **Sex ratio**: Target female-to-male ratio in groups
- **Ancestry guardrails**: Optional uploaded ancestry rules (see `checkAncestryRules`) enforced during group formation; inactive when the pedigree has no `ancestry` column. A block rule can be overridden for the session through a confirm gate requiring a stated reason; the "Ancestry" results tab reports each run's rule violations (overridden rules stay visible, marked `overridden` – see `reportAncestryViolations`) and offers the run's downloadable audit manifest

The formed groups stay in memory when a new file is read or the Pedigree Browser is trimmed. When a group then holds an animal the loaded pedigree no longer has, the Groups, Statistics and Group Detail pages say so in one sentence in place of their tables, and bringing the earlier pedigree back shows the groups again. The returned `groups` are not changed.

Up to `maxCandidates` (the "Candidates to retain" input; default 5, range 1-50) distinct candidate groupings are formed per run (issue #125); fewer are returned when the run finds fewer distinct ones. A "Candidate grouping" selector lets the user switch among them without re-running `groupAddAssign`. All reactive components below reflect the currently-selected candidate, defaulting to the best-scoring one – identical to the single-solution behavior prior to issue #125.

The results are shown on the Groups, Statistics, Group Detail and Ancestry tabs.

Value

List with reactive components:

- `groups` - List with one character vector of animal IDs per formed group; when candidates remain unplaced a final "Unused" element is appended
- `nGroups` - Number of elements of `groups`, counting the "Unused" element when present
- `groupLabels` - Character vector with one name per element of `groups`: "Group 1", "Group 2" and so on, and "Unused animals" for the final "Unused" element when there is one; `groups` itself is not named
- `score` - Optimization score from groupAddAssign (minimum group size)
- `unassigned` - Character vector of candidate IDs that appear in no element of `groups`; leftovers are collected in the trailing "Unused" element, so this is normally empty
- `groupKinship` - List of kinship matrices per group when the "Include kinship in display of groups" box is checked (default unchecked); `NULL` otherwise
- `ancestryRules` - The validated ancestry rules table loaded through the Ancestry Guardrails upload (see `checkAncestryRules`), or `NULL` when no usable file is loaded. It is the table as loaded, whether or not the pedigree has an `ancestry` column: each consumer (formation here, `modMatePairServer`) applies its own column check

See Also

modBreedingGroupsUI for the UI component

groupAddAssign for the underlying MIS algorithm

modGeneticValueServer for genetic value analysis

kinship for kinship matrix calculation

Other Shiny modules: modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modBreedingGroupsUI *Breeding Groups Module - UI Function*

Description

Breeding Groups Module - UI Function

Usage

```
modBreedingGroupsUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div containing breeding group formation UI.

References

Vinson, A. and Raboin, M.J. (2015) "A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques (*Macaca mulatta*)" *Journal of the American Association for Laboratory Animal Science*, 2015 Nov, Vol.54(6), pp.700-707.

See Also

modBreedingGroupsServer

groupAddAssign for group formation algorithm.

Other Shiny modules: modBreedingGroupsServer(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(
modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(),
modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(),
modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(),
modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(),
modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(),
modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(),
modSummaryStatsServer(), modSummaryStatsUI()

---

modCrossCenterIdentityServer

Cross-Center Identity Mapping Module - Server Function

Description

Cross-Center Identity Mapping Module - Server Function

Usage

```
modCrossCenterIdentityServer(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A list with reactive components. mergedPedigree and issues halt (via req, not NULL) until Validate has been run, and mergedPedigree also halts while issues remain:

- mergedPedigree - the resolveCrossCenterIds output, once the uploaded mapping validates clean
- issues - the checkCrossCenterMapping output (plus a synthetic type = "structural" row if a required column was missing from an upload); zero rows means clean
- confirmed - logical: has the D7 confirmation modal been accepted for the currently-validated mapping

Standalone review/export tool (D3): the merged pedigree is a downloadable artifact only – it is not written into shared$currentPedigree or any other module's reactive graph. A curator who wants the merged result to drive downstream analysis re-uploads the exported "Merged Pedigree" CSV through modInputServer's existing pedigree-file path.

See Also

modCrossCenterIdentityUI for the user interface.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modCrossCenterIdentityUI

Cross-Center Identity Mapping Module - UI Function

Description

A Shiny workflow around the script-callable checkCrossCenterMapping (Slice 1, "show every problem at once") and resolveCrossCenterIds: upload two center pedigrees and a curator-reviewed identity mapping, validate, preview the lineage changes the proposed merge would make, confirm, and export the mapping, validation results, merge summary, merged pedigree, and provenance. Retains the no-automatic-identity-inference policy – identity is established only by the uploaded mapping file, never guessed.

Usage

```
modCrossCenterIdentityUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div object containing the module's UI.

See Also

modCrossCenterIdentityServer for server logic.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modDeidentifiedExportServer` *De-Identified Export Module - Server Function*

Description

De-Identified Export Module - Server Function

Usage

```
modDeidentifiedExportServer(id, pedigree)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.
`pedigree`	reactive returning the current pedigree data.frame (`shared$currentPedigree`, D1) – not a fresh upload, unlike `modCrossCenterIdentityServer`.

Value

A list with reactive components. `exportedPedigree`, `map` and `manifest` halt (via `req`, not `NULL`) until Preview has been run:

- `exportedPedigree` - the most recent `obfuscatePed` output
- `map` - the corresponding id alias map (`obfuscatePed`'s `map=TRUE` return)
- `manifest` - the D4 transformation manifest, built from the EXACT parameters that produced `exportedPedigree` (captured at preview time, not re-read from live input state – a curator who tweaks the configuration after previewing but before exporting must not get a manifest describing different parameters than what was actually exported)
- `confirmed` - logical: has the modal confirm gate been accepted for the current preview. Resets to `FALSE` whenever the preview is regenerated (mirrors `modCrossCenterIdentityServer`'s own D5 stale-confirmation -reset pattern), so a stale confirmation can never silently unlock exports for changed output.

Downloads are not hard-gated on `confirmed` (mirroring `modCrossCenterIdentityServer`'s own precedent exactly) – per this issue's own ratified framing, "curator-controlled" means a confirmation dialog and warning text, not real access control (sec 1.2).

See Also

`modDeidentifiedExportUI` for the user interface.

`obfuscatePed` for the underlying de-identification.

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modCrossCenterIdentityServer()`, `modCrossCenterIdentityUI()`, `modDeidentifiedExportUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modMarkerGeneticsServer()`, `modMarkerGeneticsUI()`,

modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modDeidentifiedExportUI`

De-Identified Export Module - UI Function

Description

A Shiny workflow around the existing, tested de-identification primitives (obfuscatePed) and an internal manifest builder: configure export parameters, preview the de-identified output for the pedigree already loaded in the current session, confirm via a modal gate, and download 3 artifacts – the de-identified pedigree, a transformation manifest, and a distinctly labeled re-identification key.

Usage

```
modDeidentifiedExportUI(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Details

Unlike modCrossCenterIdentityUI (issue #149), this module has no `fileInput` – it reads the pedigree the curator is already working with (D1), so there is no Upload/Validate stage, only Configure & Preview, then Export (D2).

Value

A `div` object containing the module's UI.

See Also

modDeidentifiedExportServer for server logic.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modGeneticDiversityServer`

Genetic Diversity Module - Server Function

Description

Assembles the live breeding-group, genetic-value, and kinship reactive inputs into the per-group heat-map statistics (via `getGeneticDiversityStats`) and renders the red/yellow/green heat map (via `makeGeneticDiversityHeatmap`). When breeding groups have not been formed or the genetic value analysis has not been run, the module shows guidance instead of an empty plot. When a formed group holds an animal the loaded pedigree no longer has (a new file was read, or the view was trimmed, after the groups were formed), the page says so in one sentence and asks for the groups to be formed again; any other calculation error is shown once, where the heat map would be. When any animal in the groups has no birth date, a note under the heat map says how many, and how many females among them the Production and Inbreeding columns counted (those the pedigree lists as a dam) and left out.

Usage

```
modGeneticDiversityServer(
  id,
  groups,
  pedigree,
  geneticValues,
  kinshipMatrix = NULL,
  currentDate = Sys.Date(),
  kinshipFor = NULL
)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.
`groups`	reactive returning a list of character vectors of animal IDs, one per breeding group (the `groups` returned by `modBreedingGroupsServer`). That list includes the trailing "Unused" element when candidates remain unplaced, and the heat map scores it as another group. If the list is named, the names are the heat map's row labels (the app passes the names that the Breeding Groups module returns as `groupLabels`, so the last row reads "Unused animals"); an unnamed list gives "Group 1", "Group 2" and so on.
`pedigree`	reactive returning the quality-controlled pedigree data frame.
`geneticValues`	reactive returning the genetic value report data frame (with `id` and `value` columns).
`kinshipMatrix`	optional reactive returning a square kinship matrix (row and column names are animal IDs) covering every member of every group; `getGeneticDiversityStats` stops when a group member is missing from it. It is used before `kinshipFor`.
`currentDate`	Date used to derive age and the production birth window. Defaults to `Sys.Date()`.

`kinshipFor` optional function of a character vector of animal IDs that returns those animals' kinship matrix. Once groups, the pedigree and the genetic value report are all available, the module asks it for the groups' members. In the app it computes the members' kinship from them and their ancestors only, so a group may hold any animal of the pedigree, not only the Genetic Value population. With neither `kinshipMatrix` nor `kinshipFor` the module shows no heat map.

Value

A list with two reactive elements: `stats`, the per-group metric data frame (or `NULL` when data are not ready), and `heatmap`, the `ggplot` heat map (or `NULL`). Errors raised inside `getGeneticDiversityStats` (a group member missing from the kinship matrix) are not caught, so the reactives error instead of returning `NULL`.

See Also

`modGeneticDiversityUI`

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modCrossCenterIdentityServer()`, `modCrossCenterIdentityUI()`, `modDeidentifiedExportServer()`, `modDeidentifiedExportUI()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modMarkerGeneticsServer()`, `modMarkerGeneticsUI()`, `modMatePairServer()`, `modMatePairUI()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modPyramidUI()`, `modSnapshotTrendsServer()`, `modSnapshotTrendsUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

---

modGeneticDiversityUI *Genetic Diversity Module - UI Function*

Description

Genetic Diversity Module - UI Function

Usage

```
modGeneticDiversityUI(id)
```

Arguments

`id` character vector of length 1. Module namespace identifier.

Value

A `div` containing the genetic diversity heat-map UI: a housing-type selector, a guidance area, the heat-map plot, and a note about animals with no birth date (shown only when there are any).

See Also

modGeneticDiversityServer

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modGeneticValueServer *Genetic Value Analysis Module - Server Function*

Description

Genetic Value Analysis Module - Server Function

Usage

```
modGeneticValueServer(
  id,
  pedigree,
  speciesOverrides = reactive(NULL),
  twinRelations = reactive(NULL)
)
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigree	reactive returning pedigree data frame.
speciesOverrides	reactive returning the user-configurable species overrides loaded at boot by loadSpeciesOverrides (a list with breedingTable, gestationTable, breedingAgeDefault, gestationDefault), or NULL. Threaded into reportGV. Defaults to reactive(NULL) so no config file means bundled behavior.
twinRelations	reactive returning a validated twin/zygosity sidecar data.frame (id1, id2, code), or NULL. Unlike the kinship-override file (uploaded through this module's own UI), this data does not originate inside this module – it is uploaded on the Pedigree Browser's Diagram tab and threaded in by appServer (BL-N Slice 3), so a declared MZ-twin pair's corrected kinship is reflected in reportGV's output regardless of which tab the user visits first. Defaults to reactive(NULL) (no twins declared).

Value

List with `geneticValues`, `topAnimals`, `nAnalyzed`, `kinshipMatrix`, `kinshipOverrides` (the validated kinship-override data frame loaded through this module's kinship-override upload, or `NULL` when none is loaded or it cannot be read), `founderStats`, `maleFounders`, `femaleFounders`, and `snapshotSource` (issue #167 Slice 4: the most recent run's analyzed pedigree, its `nprcgenekeeprGV` object, and its `guIter`/`guThresh`, captured atomically for `modSnapshotTrendsServer`, plus `reportArgs` (issue #170 Slice 3): the other nine settings the run passed to `reportGV` – the species tables and their defaults, the kinship overrides, the twin pairs and the three ranking cutoffs – so the rows built for earlier dates match the analysis beside them).

References

Lacy, R.C. (1989) *Zoo Biology*, **8**, 111-123.

See Also

`modGeneticValueUI`

`modBreedingGroupsServer` for using results.

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modCrossCenterIdentityServer()`, `modCrossCenterIdentityUI()`, `modDeidentifiedExportServer()`, `modDeidentifiedExportUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modMarkerGeneticsServer()`, `modMarkerGeneticsUI()`, `modMatePairServer()`, `modMatePairUI()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modPyramidUI()`, `modSnapshotTrendsServer()`, `modSnapshotTrendsUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

---

modGeneticValueUI *Genetic Value Analysis Module - UI Function*

Description

Genetic Value Analysis Module - UI Function

Usage

```
modGeneticValueUI(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Value

A `div` containing genetic value analysis UI.

References

Vinson, A. and Raboin, M.J. (2015) *Journal of the American Association for Laboratory Animal Science*, **54**(6), 700-707.

See Also

modGeneticValueServer

geneDrop for gene dropping simulation.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modGvAndBgDescServer *Genetic Value and Breeding Group Description Module - Server Function*

Description

Server logic for genetic value and breeding group description module. This module is primarily informational and does not require reactive logic.

Usage

```
modGvAndBgDescServer(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

NULL (no reactive outputs).

See Also

modGvAndBgDescUI for the user interface.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(),

modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modGvAndBgDescUI` *Genetic Value and Breeding Group Description Module - UI Function*

Description

Creates user interface displaying detailed documentation about genetic value analysis and breeding group formation algorithms.

Usage

```
modGvAndBgDescUI(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Value

A `div` object containing the description HTML.

See Also

modGvAndBgDescServer for server logic.

modGeneticValueUI for genetic value analysis.

modBreedingGroupsUI for breeding group formation.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modInputServer` *Data Input and Quality Control Module - Server Function*

Description

Server logic for data input module handling file uploads, parsing of pedigree and genotype data files, and quality control validation.

Usage

```
modInputServer(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Value

A list with reactive components:

- `cleanedStudbook` - The QC-cleaned studbook data
- `genotypeData` - Genotype data (`id`, `first`, `second`) taken from the cleaned studbook when it carries genotypes. A separately uploaded genotype file that cannot be read, or that fails `checkGenotypeFile`, is not used, leaving `NULL`; the QC Summary says why
- `qcSummary` - Summary of QC results: a list with `errors`, `warnings` and `records` (counts of error rows, warning rows and cleaned records)
- `minSireAge` - The minimum sire age floor (numeric, or `NULL` to use the species+sex breeding-age table default). It is re-parsed from the live text box, so it can differ from the value the last "Read and Check Pedigree" used if the box was edited afterward
- `minDamAge` - The minimum dam age floor (numeric, or `NULL` to use the species+sex breeding-age table default); the same live-box caveat as `minSireAge` applies
- `isReady` - Logical indicating if data is ready for next step
- `debugMode` - Logical reflecting the Input tab's "Debug on" checkbox
- `changedCols` - Renamed/changed-column diagnostics from QC
- `errorLst` - The QC error list, used for dynamic tab management
- `pedigreeFileName` - The uploaded file's name, used for dynamic tab management

Note

(Developer note) This module is the reference implementation of the package's internal Shiny module contract – see `docs/architecture/module-contract.md` in the package source (a developer-only file; it does not ship with the installed package).

See Also

modInputUI for the user interface.

modPedigreeServer for using the cleaned data.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modInputUI *Data Input and Quality Control Module - UI Function*

Description

Creates user interface for data input including file uploads for pedigree and genotype data with various format options, followed by quality control validation. The File Content choices are a pedigree file only, pedigree and genotypes in one file, pedigree and genotypes in separate files, and "Focal animals only; pedigree built from database". Optional minimum sire and dam age fields and a "Debug on" box sit with the file options; the results appear on the QC Summary, Errors, Warnings and Cleaned Data tabs.

Usage

```
modInputUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div object containing the data input UI.

See Also

modInputServer for server logic.

modPedigreeUI for pedigree browsing after QC.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(),

modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modMarkerGeneticsServer

Marker Genetics Module - Server Function

Description

Reads an uploaded long-format marker genotype file (D1 format: id, locus, allele1, allele2), validates and pivots it (checkSequenceGenotypeFile, which applies the checkMarkerGenotypeFile checks plus a literal-"." rejection and a maxLoci warning, then buildMarkerGenotypeMatrix; the Center B upload is validated the same way), estimates marker-based kinship independent of pedigree (markerKinship), and surfaces a per-animal comparison of pedigree-based mean kinship (indivMeanKin, from the pedigree kinship given by kinshipMatrix or kinshipFor) alongside the new marker-based mean kinship (markerMeanKin) – an independent check on the pedigree-implied relatedness, not a replacement for it. A second tab surfaces the heterozygosity diagnostic: per-animal observed heterozygosity (markerObservedHeterozygosity) alongside population-level expected heterozygosity (markerExpectedHeterozygosity). A third tab surfaces the Mendelian-exclusion parentage diagnostic (markerParentageExclusion): the pedigree's recorded dam/sire cross-referenced against the uploaded genotypes, flagging any recorded parent the genotype evidence contradicts. A fourth tab, "Cross- Center", surfaces a between-population differentiation statistic (markerFst) between the first uploaded file (implicitly "Center A") and a second, independently uploaded Center B genotype file – a population-level, two-dataset comparison, unrelated to the per-animal cross-center identity linking of resolveCrossCenterIds (Slice 4). A fifth tab, "Candidate Parent Assignment" (issue #147 Slice 2), surfaces markerParentageLikelihood: for every (offspring, role) pair the Parentage Exclusion tab's own diagnostic flags as Mendelian -inconsistent, it ranks candidate replacement parents by a CERVUS-style multilocus likelihood (LOD) score. This tab needs no new file input – it reads the same uploaded genotype file and pedigree already wired to the other tabs – and is report-only, matching the Parentage Exclusion tab's own precedent: it never writes to pedigree. Candidates come from getPotentialParents, which needs the pedigree's fromCenter column; without it the table is blank, and the tab says so above the table.

Usage

```
modMarkerGeneticsServer(id, kinshipMatrix = NULL, pedigree, kinshipFor = NULL)
```

Arguments

id	character vector of length 1. Module namespace identifier.
kinshipMatrix	optional reactive returning the full pedigree-based kinship matrix (row and column names are animal IDs), or NULL while upstream analysis has not yet been run. It is used before kinshipFor.
pedigree	reactive returning the current pedigree data frame (columns id, sire, dam), or NULL while upstream analysis has not yet been run.

`kinshipFor` optional function of a character vector of animal IDs that returns those animals' kinship matrix; with no IDs it returns every animal's. The Kinship Comparison asks it for every animal once a genotype file is uploaded and keeps only the means; the realized relatedness table asks it for every animal once the pedigree and the two inputs are usable. `appServer` passes one that applies the twins and overrides. With neither `kinshipMatrix` nor `kinshipFor`, `indivMeanKin` is `NA` and there is no realized relatedness table. `NULL` (the default) is not used.

Details

A sixth tab, "Linkage and LD Block Metrics" (issue #153 Slice 5), wires in three additional analyses. A locus-metadata file (`locus`, `chrom`, `pos`, optionally `cM`) is validated and classified into a three-tier coverage report (`checkLocusMetadata`, D2). The realized-relatedness-variance table (`markerRealizedRelatednessVariance`, D3a) needs only the existing `pedigree` and its kinship plus a curator-supplied chromosome count and genetic-map length – no genotype file at all. The LD-block table (`markerLdBlock`, D3b) reads its OWN, dedicated `linkageGenotypeFile` upload – deliberately independent of the other five tabs' shared `genotypeFile`, since Shiny renders every `tabPanel`'s output bindings regardless of which tab is visible: a multiallelic file uploaded through the shared input would break the other five tabs' own DT outputs simultaneously, not just this tab's. Validated through the multiallelic-tolerant sibling validator (`checkLinkageMarkerGenotypeFile`) rather than `checkSequenceGenotypeFile`, the validator for the shared upload. Any exported LD-block table is de-identified (`obfuscateLdBlocks`) behind a curator confirm-gate reusing `modDeidentifiedExportServer`'s tested Generate-Preview -> Confirm -> Confirm-OK pattern (D9).

A seventh tab, "Genomic ROH (F_ROH)" (issue #152 Slice 5), computes `computeGenomicROH` from the shared genotype and locus-metadata uploads, with the same confirm-gated, de-identified export pattern (genotype matrix, F_ROH table, manifest).

An eighth tab, "MHC Haplotype Reporting" (issue #148 Slice 4), reads its OWN dedicated `mhcHaplotypeFile` upload (validated by `checkMhcHaplotypeFile`) and reports `mhcHaplotypeFrequency`'s per-haplotype summary and `mhcHaplotypeCarriers`'s rare-haplotype carrier list at the two rarity thresholds shown next to the tables, with a persistent descriptive-only caveat, the call counts and frequency denominator, and (when `pedigree` is available) how many pedigree animals have a designation. Its export (summary, carrier list aliased through `obfuscateMhcHaplotypes`, and a manifest) goes through the same confirm gate. The alias map covers pedigree animals only, so the export is not generated – with the reason shown – while any animal in the MHC file is absent from the loaded pedigree.

This module never touches the existing single-locus genotype path (`checkGenotypeFile`/`addGenotype`/`hasGenotype`/`getGVGenotype`/`geneDrop`) – the D1 long-format schema is a new, sibling concern.

Value

A named list of reactive elements: `markerGenotype`, the raw uploaded genotype data frame (or `NULL` before upload); `markerKinshipMatrix`, the marker-based `id` x `id` kinship matrix (or `NULL`); `comparisonTable`, the per-animal `indivMeanKin`/`markerMeanKin` comparison data frame (or `NULL`; `indivMeanKin` is `NA` for every row when the pedigree kinship matrix is `NULL` or errors, and for any genotyped id absent from it); `heterozygosityTable`, the per-animal `ho`/`he` heterozygosity data frame (`he` is the population-wide mean expected heterozygosity, repeated per row) (or `NULL`); `exclusionTable`, the `markerParentageExclusion` data frame: one row per recorded dam/sire

pair with exclusionCount, nLoci and a flagged column (TRUE where the count exceeds the tolerance), so pairs that are not flagged are included (or NULL before a genotype file and a pedigree are both available); crossCenterGenotypeB, the raw uploaded Center B genotype data frame (or NULL before upload); crossCenterTable, the markerFst locus/fst data frame with a trailing "Pooled" row (or NULL before both center files are uploaded); candidateAssignmentTable, the markerParentageLikelihood ranked-candidate data frame (a zero-row, full-column-shape data frame when no pair is flagged; or NULL before a genotype file and a pedigree are both available); isReady, TRUE once comparisonTable has a value, and FALSE (never an error) when it has none, including when the uploaded genotype file cannot be used; locusMetadataTable, the checkLocusMetadata output (or NULL before a locus-metadata file is uploaded); realizedRelatednessTable, the markerRealizedRelatednessVariance output (or NULL before the pedigree and its kinship are both available); ldBlockTable, the markerLdBlock output (or NULL before a genotype file and a locus-metadata file are both uploaded, or before a pedigree is available if the founders-only restriction is checked); ldBlockExportTable, the obfuscateLdBlocks-de-identified export preview (or NULL before "Generate De-Identified Export Preview" is clicked with both ldBlockTable and pedigree available); ldBlockExportConfirmed, FALSE until the confirm-gate modal's own Confirm button is clicked for the current export preview; sequenceRohTable, the computeGenomicROH output (or NULL before a genotype file and a locus-metadata file are both uploaded, or while a threshold input is invalid); sequenceExportGenotypeMatrix, sequenceExportRohTable and sequenceExportManifest, the de-identified genotype matrix, de-identified F_ROH table and export manifest captured at "Generate De-Identified Export Preview" (each NULL before then, and when any genotype id is absent from the pedigree or the ROH table or pedigree is NULL); sequenceExportConfirmed, FALSE until that export's confirm-gate modal is accepted for the current preview; mhcHaplotypeSummaryTable, the mhcHaplotypeFrequency summary data frame (or NULL before an MHC haplotype file is uploaded, or while a rarity threshold is invalid); mhcHaplotypeCarrierTable, the mhcHaplotypeCarriers rare-haplotype carrier data frame (same NULL conditions); mhcExportTables, a list of the summary, de-identified carriers and manifest data frames captured at "Generate De-Identified Export Preview" (or NULL before then, without a pedigree, or while any MHC-file animal is absent from the pedigree); and mhcExportConfirmed, FALSE until the MHC export's confirm-gate modal is accepted for the current preview.

See Also

modMarkerGeneticsUI

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modMarkerGeneticsUI *Marker Genetics Module - UI Function*

Description

Marker Genetics Module - UI Function

Usage

```
modMarkerGeneticsUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div containing the marker-genetics UI: four file upload controls (marker genotype, Center B genotype for the Cross-Center comparison, multiallelic-tolerant linkage genotype, and locus metadata), a guidance area, and a tab set with the Kinship Comparison, Heterozygosity, Parentage Exclusion, Cross-Center, Candidate Parent Assignment, Linkage and LD Block Metrics, Genomic ROH (F_ROH), and MHC Haplotype Reporting sub-tabs. The last three sub-tabs carry their own controls and confirm-gated de-identified exports; the MHC sub-tab also has its own haplotype-file upload and rarity-threshold inputs.

See Also

modMarkerGeneticsServer

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modMatePairServer *Mate Pair Analysis Module - Server Function*

Description

Reports eligible individual mate-pair candidates via reportMatePairs (issue #151 Slice 1), wrapped in a curator-facing configuration panel for the D4-ratified population scope (populationSource: "allAlive" – the animals that are alive: a status of ALIVE when the pedigree has a status column, otherwise no recorded exit date, and every id when it has neither column (the rule Breeding Groups uses); "topRanked" – the top nTopAnimals ids in geneticValues' own report order, mirroring modBreedingGroupsServer's own topRanked reading; "custom" – a pasted, delimiter-separated id list), the D2 minimum- age floor, and the D5-ratified exclude-list textarea. Kept structurally and file-wise separate from modBreedingGroupsServer (D1) – this module shares no code with it.

Usage

```
modMatePairServer(
  id,
  pedigree,
  kinshipMatrix = NULL,
  markerKinshipMatrix,
  geneticValues,
  ancestryRules = NULL,
  kinshipFor = NULL
)
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigree	reactive returning the current pedigree data frame (columns id, sire, dam, sex, age, optionally exit).
kinshipMatrix	optional reactive returning a pedigree-based kinship matrix (row/column names are animal IDs) covering the population. It is used before kinshipFor.
markerKinshipMatrix	reactive returning the genotype-only KING- robust kinship matrix from modMarkerGeneticsServer's own markerKinshipMatrix return element, or NULL before a genotype file has been uploaded.
geneticValues	reactive returning the current genetic-value report data.frame (shared$geneticValues, with id, indivMeanKin, gu columns), or NULL before the Genetic Value Analysis tab has been run.
ancestryRules	optional reactive returning the validated ancestry rules table (see checkAncestryRules) from modBreedingGroupsServer's ancestryRules return element, or NULL when no rules are loaded. NULL (the default, or a reactive that returns NULL) applies no rules, and the results are identical to a run before this argument existed.
kinshipFor	optional function of a character vector of animal IDs that returns those animals' kinship matrix. At each "Find Eligible Pairs" click the module asks it for the population. appServer passes one that computes kinship from those animals and their ancestors only. With neither kinshipMatrix nor kinshipFor the click does nothing.

Details

The geneticValues wiring detail. shared$geneticValues (as threaded from appServer.R, matching every other module's own convention) is the flat reportGV()$report data.frame, not the list(report = ...) shape reportMatePairs itself expects. This server wraps it (list(report = geneticValues())) immediately before calling reportMatePairs() – omitting the wrap would not error (a data.frame's $report accessor returns NULL, not a condition), it would silently leave every genetic- value column NA.

Population scoping is a hard dependency, not a fallback-recompute. Unlike modBreedingGroupsServer's optional kinshipMatrix (which recomputes from pedigree when absent), this module always receives its kinship from its caller – a kinshipMatrix, or the kinshipFor function appServer.R

passes, asked for the population at each run – and simply depends on it (module-contract rule 5: upstream absence is `req()`), matching `modMarkerGeneticsServer`'s own simpler precedent – there is no standalone use case for this module that would need an independent recompute path.

Ancestry guardrails (issue #169). `ancestryRules` carries the rules loaded on the Breeding Groups tab. When rules are loaded and the pedigree has an `ancestry` column, each run hands them to `reportMatePairs`: a pair matching a `block` rule moves to the Excluded tab (reason `"ancestry rule"`) and a pair matching a `flag` rule stays in Eligible Pairs, annotated. The rules are read once, at the "Find Eligible Pairs" click, so loading or clearing rules afterwards never rewrites a finished run's tables. The module checks for the `ancestry` column before passing rules (the report function stops without it); a rules-loaded run on a pedigree with no such column runs exactly as it does with no rules, and the status line says the guardrails are inactive. An always-visible status line reports the loaded state.

Overriding a block rule. With active rules, the "Ancestry Guardrails" section lists the block rules not yet overridden. "Override rule..." opens a confirmation step that shows the warning text verbatim and requires a written reason (a blank reason is refused); confirming relaxes that one rule for this tab's runs, for the session, until the overrides are cleared or the rules change. An overridden rule's pairs stay in Eligible Pairs marked `"overridden"` (severity still `"block"`); they are never silently dropped. Each override is recorded per tab – Breeding Groups keeps its own – and the next run applies it; a finished run is never rewritten.

The Ancestry tab. Shows the displayed run's coverage summary (how many candidate animals carry each ancestry classification and which classifications no rule reaches) and a "Download Audit Manifest" button. The manifest is one row per rule: the rule as written, whether it was overridden and why, how many otherwise-eligible pairs it matched, the animal census by classification, and the confirmation warning verbatim. It describes the displayed run's own rules and overrides, and is unavailable for a run that had no rules in effect.

Value

A list with three reactive elements: `pairs`, the eligible- pairs data.frame from the most recent `reportMatePairs()` run (see that function's own return documentation for columns); `excluded`, the corresponding excluded-pairs data.frame; and `isReady`, `TRUE` once a run has completed. Before the first run `pairs()` and `excluded()` halt (via `req`) rather than returning `NULL`.

See Also

`modMatePairUI`

`reportMatePairs`

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modCrossCenterIdentityServer()`, `modCrossCenterIdentityUI()`, `modDeidentifiedExportServer()`, `modDeidentifiedExportUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modMarkerGeneticsServer()`, `modMarkerGeneticsUI()`, `modMatePairUI()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modPyramidUI()`, `modSnapshotTrendsServer()`, `modSnapshotTrendsUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

modMatePairUI *Mate Pair Analysis Module - UI Function*

Description

Mate Pair Analysis Module - UI Function

Usage

```
modMatePairUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A `div` containing the mate-pair analysis UI: a candidate- population/age/exclude-list configuration panel (with a collapsed "Ancestry Guardrails" section – holding the per-rule override controls – and an always-visible ancestry-rules status line) and an "Eligible Pairs" / "Excluded" / "Ancestry" tabbed result view (the last holding the coverage summary and the audit-manifest download).

See Also

`modMatePairServer`

`reportMatePairs` for the underlying report function.

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modCrossCenterIdentityServer()`, `modCrossCenterIdentityUI()`, `modDeidentifiedExportServer()`, `modDeidentifiedExportUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modMarkerGeneticsServer()`, `modMarkerGeneticsUI()`, `modMatePairServer()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modPyramidUI()`, `modSnapshotTrendsServer()`, `modSnapshotTrendsUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

modORIPReportingServer *ORIP Reporting Module - Server Function*

Description

Server logic for ORIP reporting module. Generates summary statistics and formatted reports for Office of Research Infrastructure Programs submissions.

Usage

```
modORIPReportingServer(
  id,
  pedigree = NULL,
  geneticValues = NULL,
  siteConfig = NULL
)
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigree	reactive returning pedigree data frame.
geneticValues	reactive returning the flat genetic value report data frame (with indivMeanKin and gu columns), not a list(report =).
siteConfig	reactive returning site configuration from getSiteInfo(), with center, nodename, user, sysname and release. When NULL, the module falls back to getSiteInfo(expectConfigFile = FALSE).

Value

A list with one reactive component, colonySummary, which returns list(nTotal, nMales, nFemales, nFounders).

The tab is mounted only when shouldShowOripTab is TRUE (the ONPRC site configuration; see appUI).

See Also

modORIPReportingUI for the user interface

getSiteInfo for site configuration

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modORIPReportingUI *ORIP Reporting Module - UI Function*

Description

Creates user interface for ORIP (Office of Research Infrastructure Programs) reporting. The module is under development and is intended to grow into formatted reports suitable for submission to ORIP as part of primate center grant reporting requirements.

Usage

```
modORIPReportingUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Details

The ORIP Reporting tab currently shows:

- A site information table
- A colony summary (animal, sex and founder counts)
- Mean kinship and mean genome uniqueness

with two CSV exports. Breeding-program statistics, founder analysis and formatted reports are not implemented. The tab is mounted only when `shouldShowOripTab` is `TRUE` (the ONPRC site configuration; see `appUI`).

Value

A `div` object containing the ORIP reporting UI.

See Also

`modORIPReportingServer` for server logic.

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modCrossCenterIdentityServer()`, `modCrossCenterIdentityUI()`, `modDeidentifiedExportServer()`, `modDeidentifiedExportUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modMarkerGeneticsServer()`, `modMarkerGeneticsUI()`, `modMatePairServer()`, `modMatePairUI()`, `modORIPReportingServer()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modPyramidUI()`, `modSnapshotTrendsServer()`, `modSnapshotTrendsUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

---

modPedigreeServer *Pedigree Browser Module - Server Function*

Description

Server logic for pedigree browser module handling focal animal selection (typed, pasted or uploaded, or set by clicking a diagram node), pedigree processing, filtering (focal trim and the Display Unknown IDs filter), the Diagram tab (edge style, names, twin connectors, PNG export), the twin/zygosity sidecar upload, and data export.

Usage

```
modPedigreeServer(id, studbook)
```

Arguments

id	character vector of length 1. Module namespace identifier.
studbook	reactive returning the cleaned studbook data from modInput.

Details

This module processes the studbook by:

- Adding a population column via setPopulation(): the focal animals, or the living animals when none are entered
- Adding a pedNum column via findPedigreeNumber()
- Ensuring a gen column exists via findGeneration()
- Optionally trimming to the ancestors and descendants of focal animals via trimPedigree() and getDescendantPedigree()

Value

A list of reactive values:

- pedigree - Filtered pedigree for display (respects trim/unknown settings)
- analysisPedigree - The pedigree the other tabs analyze (focal-animal trim applied, no Display Unknown IDs filter, which affects only the table); appServer uses it, not pedigree, as the shared current pedigree
- processedPedigree - Full pedigree with population, pedNum, gen columns
- focalAnimals - Character vector of focal animal IDs
- nAnimals - Count of animals in filtered pedigree
- populationCount - Count of animals marked as population
- isReady - Logical indicating if pedigree data is ready. With no studbook loaded, isReady() and nAnimals() halt with a silent req error instead of returning FALSE and 0
- twinRelations - The validated twin/zygosity sidecar (NULL if none uploaded, if invalid, or if no pair has both animals in analysisPedigree). Checked against the whole studbook, not the trimmed view: an id that is not in the studbook, or any other failure of checkTwinRelations(), shows an error notification and makes the whole sidecar NULL; a valid pair with an animal outside analysisPedigree (a focal-animal trim) is left out of this value, with no message. Unlike the Diagram tab's own rendering, this is the raw, ungated reactive – not filtered by the "Show Twin Connectors" toggle – so callers outside this module (e.g. appServer) see the validated data regardless of that toggle's state (BL-N Slice 3).

See Also

modPedigreeUI for the UI component

setPopulation for population marking

trimPedigree for pedigree trimming

findPedigreeNumber for pedigree numbering

findGeneration for generation calculation

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modPedigreeUI` *Pedigree Browser Module - UI Function*

Description

Creates user interface for browsing and filtering pedigree data, including focal animal selection (typed or pasted IDs, or an uploaded file), trimming options, the Display Unknown IDs filter, and export. A Diagram tab draws the pedigree (rectilinear or direct edge style, optional names and twin connectors, PNG export, and click-to-focal), with a twin/zygosity sidecar upload; the diagram is drawn only up to 400 animals (rectilinear) or 750 animals (direct).

Usage

```
modPedigreeUI(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Value

A `div` object containing pedigree browser UI.

See Also

`modPedigreeServer` for server logic.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

modPotentialParentsServer

Potential Parents Module - Server Function

Description

Server logic for the Potential Parents module. On button press, it calls `getPotentialParents` against the current pedigree, flattens the result into a sortable table, and exposes it for CSV download. The surface degrades gracefully when the pedigree reactive returns `NULL` or zero rows, when the pedigree lacks the `fromCenter` colony-origin field, or when no in-colony animal has an unknown parent.

Usage

```
modPotentialParentsServer(
  id,
  pedigree = NULL,
  minSireAge = NULL,
  minDamAge = NULL,
  gestationTable = NULL,
  gestationDefault = NULL
)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.
`pedigree`	reactive returning the current pedigree data.frame. The default `NULL` is not usable: the module calls `pedigree()`, so pass a reactive (which may itself return `NULL`).
`minSireAge`	minimum age in years for a male to be proposed as a sire. May be a plain numeric, `NULL`, or a reactive returning either. `NULL` (the default) uses the species- and sex-specific breeding-age table default via `getPotentialParents`.
`minDamAge`	minimum age in years for a female to be proposed as a dam. Same forms and default as `minSireAge`, applied to females.
`gestationTable`	optional species-to-gestation lookup passed to `getSpeciesGestation` when defaulting the gestation window; `NULL` (the default) uses the bundled `speciesGestation` table. Supplied at boot from the user-configurable species overrides, so a colony's CSV values drive the prefill default.
`gestationDefault`	optional integer fallback (days) for a pedigree whose species is absent from `gestationTable`, passed through to the gestation prefill; `NULL` (the default) keeps the built-in 210. Supplied at boot from the user-configurable species overrides.

Value

A list of reactive expressions:

- potentialParents - the raw getPotentialParents result (or NULL).
- tableData - the flattened results data.frame.
- gestationDefault - the species-keyed default gestation window (days) used to prefill the maximum-gestational-period input.

See Also

modPotentialParentsUI for the user interface.

getPotentialParents for the underlying computation.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modPotentialParentsUI *Potential Parents Module - UI Function*

Description

Creates the user interface for identifying potential parents of in-colony animals that have at least one unknown parent. The user sets a maximum gestational period, presses a button to compute candidate sires and dams on the current pedigree, views a sortable results table, and downloads the results as CSV.

Usage

```
modPotentialParentsUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div object containing the Potential Parents UI.

See Also

modPotentialParentsServer for server logic.

getPotentialParents for the underlying computation.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modPyramidServer *Age-Sex Pyramid Module - Server Function*

Description

Age-Sex Pyramid Module - Server Function

Usage

```
modPyramidServer(id, pedigreeData)
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigreeData	reactive returning pedigree data frame.

Value

A list with two elements: pedigree, a reactive returning the pedigree data frame, and animalCount, a reactive returning the number of animal rows.

See Also

modPyramidUI

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

modPyramidUI *Age-Sex Pyramid Module - UI Function*

Description

Age-Sex Pyramid Module - UI Function

Usage

```
modPyramidUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A `div` containing age-sex pyramid UI.

See Also

`modPyramidServer`

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modCrossCenterIdentityServer()`, `modCrossCenterIdentityUI()`, `modDeidentifiedExportServer()`, `modDeidentifiedExportUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modMarkerGeneticsServer()`, `modMarkerGeneticsUI()`, `modMatePairServer()`, `modMatePairUI()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modSnapshotTrendsServer()`, `modSnapshotTrendsUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

modSnapshotTrendsServer *Genetic-Health Trends Module - Server Function*

Description

Genetic-Health Trends Module - Server Function

Usage

```
modSnapshotTrendsServer(id, snapshotSource, fullPedigree = reactive(NULL))
```

Arguments

id	character vector of length 1. Module namespace identifier.
snapshotSource	reactive returning list(ped, geneticValue, guIter, guThresh) from the most recent Genetic Value Analysis run (see modGeneticValueServer's snapshotSource return element), or erroring (shiny req() semantics) before any run has happened.
fullPedigree	reactive returning the full quality-controlled pedigree (the Pedigree tab's processedPedigree, never the focal-trimmed analysisPedigree, which would shrink every past population), or erroring (shiny req() semantics) before a studbook is loaded. Feeds the "Build Earlier Snapshots" box. Defaults to reactive(NULL): no pedigree, so the box waits.

Value

A list with reactive components:

- history - the current validated snapshot history (or NULL before any upload or generation).
- deltas - the current delta comparison (see calcSnapshotDeltas), keyed off the sidebar's rule/from/ to selectors.
- isReady - logical: TRUE once history is non-NULL (a history has been uploaded or generated).
- backfillSupport - the table of what each date of the default earlier-snapshot series rests on (previewBackfillSupport of fullPedigree()), independent of the chosen start year; erroring (req()) until a pedigree is loaded.

The "Build Earlier Snapshots" button calls backfillColonySnapshots with the full pedigree, the guIter, guThresh and reportArgs of snapshotSource(), the chosen start year and the history already held, and adds nothing of its own to the numbers. It does nothing before an analysis has run. Warnings from the build are summarized in one notification; a failure is reported and leaves the history as it was.

The generated snapshot's membershipRule is auto-derived from snapshotSource()pedpopulation (issue #167 Slice 4, S760 owner decision): "focalPopulation" when the analyzed pedigree's population column excludes any animal, "wholePedigree" otherwise – truthful by construction from what reportGV() actually analyzed, never a user-set dropdown that could contradict the data and hit createColonySnapshot's stop(). The delta comparison keeps a user-facing rule selector, populated from the uploaded/generated history's own membershipRule values.

A checkSnapshotHistory violation on a malformed history upload surfaces as a notification (module-contract rule 5: this is not the same seam as inter-module malleability – the specific violation is shown, not disguised as "no data yet") and leaves the current history unchanged.

See Also

modSnapshotTrendsUI for the user interface.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(),

modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modSnapshotTrendsUI` *Genetic-Health Trends Module - UI Function*

Description

The new top-level Genetic-Health Trends tab (issue #167 Slice 4, D7): a Shiny workflow around the longitudinal colony-snapshot functions already shipped in Slices 1-3 (checkSnapshotHistory/readSnapshotHistory/appendColonySnapshot, createColonySnapshot, calcSnapshotDeltas/plotSnapshotTrends). Upload or grow a snapshot-history CSV, generate a new dated snapshot from the analysis already loaded on the Genetic Value Analysis tab, review trend plots and a per-metric delta comparison (with the D4 comparability flag visible), and download both the updated history and the delta table. A "Build Earlier Snapshots" box rebuilds past 1 Januaries from the studbook's birth and exit dates (issue #170, backfillColonySnapshots), and an "Earlier Snapshots" tab shows what each of those dates rests on (previewBackfillSupport).

Usage

```
modSnapshotTrendsUI(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Value

A `div` object containing the module's UI.

See Also

modSnapshotTrendsServer for server logic.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSummaryStatsServer(), modSummaryStatsUI()

`modSummaryStatsServer` *Summary Statistics Module - Server Function*

Description

Server logic for summary statistics module displaying genetic analysis results including kinship statistics, histograms, box plots, and relationship designation analysis.

Usage

```
modSummaryStatsServer(
  id,
  geneticValues,
  pedigree,
  kinshipMatrix = NULL,
  founderStats = NULL,
  kinshipOverrides = NULL,
  twinRelations = NULL,
  kinshipFor = NULL
)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.
`geneticValues`	reactive returning genetic value analysis results. Must be a data frame with columns `indivMeanKin` and `gu` (the `id` column is not read). Optional `zScores` column (what `reportGV()` emits; the legacy name `zScore` is also accepted) enables z-score plots.
`pedigree`	reactive returning pedigree data frame with columns `id`, `sire`, `dam`, and `sex`. Optionally `gen`.
`kinshipMatrix`	optional reactive returning kinship matrix. When supplied it is used unchanged, so it should already carry any twin correction and kinship overrides. If NULL, or if the reactive errors or returns NULL, the module asks `kinshipFor`, then calculates kinship from the pedigree.
`founderStats`	optional reactive returning a list of founder statistics (`fe`, `fg`, `total`, `nMaleFounders`, `nFemaleFounders`). When supplied, a founder summary table is rendered on the Summary Statistics tab (monolith parity). If NULL, it is omitted.
`kinshipOverrides`	optional reactive returning a validated outside-information kinship-override data frame (`id1`, `id2`, `kinship`); see `applyKinshipOverrides`. When the module recomputes kinship from the pedigree (the fallback; in the app `kinshipFor` already applies them), the overrides are applied to that matrix, so the relationship table and the kinship CSV export reflect the supplied values regardless of tab order. The override moves the kinship *value* only; the `relation` *label* stays pedigree-derived (it is computed from pedigree structure, not from the kinship

value). Overridden pairs are flagged with a logical `overridden` column in the relationship table. `NULL` (the default) is a no-op.

`twinRelations` optional reactive returning a validated twin/zygosity sidecar data.frame (`id1`, `id2`, `code`); see `checkTwinRelations`. When the module recomputes kinship from the pedigree (the fallback; in the app `kinshipFor` already reflects it), it is passed straight through to `kinship` so the relationship table and the kinship CSV export reflect a declared MZ-twin pair's corrected identity regardless of tab order (BL-N Slice 3). `NULL` (the default) is a no-op.

`kinshipFor` optional function of a character vector of animal IDs that returns those animals' kinship matrix; with no IDs it returns every animal's. The kinship CSV download and the relationships download ask it for every animal at each click, and the relationship classes download asks once and keeps only its class counts, so no kinship matrix or relationship table is kept between clicks. `appServer` passes one that applies the twins and overrides. A provided `kinshipMatrix` is used first; with neither, the module calculates kinship from the pedigree. `NULL` (the default) is not used.

Details

This module provides:

- Summary statistics (counts, mean kinship, genome uniqueness), a six-number summary with skewness and kurtosis, a founder table and an Effective Population Size block
- Histograms and box plots for genetic value distributions, each exportable as PNG (six plots)
- Relationship classification using `convertRelationships()`
- Relationship class frequency tables using `makeRelationClassesTable()`
- First-order relative counts using `countFirstOrder()`
- CSV export for the kinship matrix, founders, relationships, the first-order counts and the relationship class table

Value

A list with reactive components:

- `summaryData` - Summary statistics (nAnimals, meanMK, meanGU)
- `relationships` - Pairwise relationship designations from `convertRelationships()`. When `kinshipOverrides` are supplied, a logical `overridden` column flags the pairs whose kinship value came from an override.
- `relationClasses` - Relationship class frequency table from `makeRelationClassesTable()`
- `firstOrderCounts` - First-order relative counts per animal from `countFirstOrder()`
- `mkSummary` - Six-number summary of mean kinship
- `guSummary` - Six-number summary of genome uniqueness
- `mkShape`, `guShape` - Shape statistics (skewness and kurtosis) of mean kinship and genome uniqueness
- `mkHistogram`, `zscoreHistogram`, `guHistogram` - The histogram plots (`ggplot` objects)
- `meanKinshipBoxPlot`, `zscoreBoxPlot`, `guBoxPlot` - The box plots (`ggplot` objects)

See Also

modSummaryStatsUI for the user interface

convertRelationships for relationship classification

makeRelationClassesTable for relationship class summary

countFirstOrder for first-order relative counting

kinship for kinship matrix calculation

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsUI()

---

modSummaryStatsUI *Summary Statistics Module - UI Function*

Description

Creates user interface for summary statistics display including histograms and box plots for mean kinship, z-scores, and genome uniqueness.

Usage

```
modSummaryStatsUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div object containing summary statistics UI.

See Also

modSummaryStatsServer for server logic.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modCrossCenterIdentityServer(), modCrossCenterIdentityUI(), modDeidentifiedExportServer(), modDeidentifiedExportUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modMarkerGeneticsServer(), modMarkerGeneticsUI(), modMatePairServer(), modMatePairUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSnapshotTrendsServer(), modSnapshotTrendsUI(), modSummaryStatsServer()

obfuscateDate *Obfuscate dates with a random day offset*

Description

Get the baseDate and add a random number of days taken from a uniform distribution bounded by -maxDelta and maxDelta. Insure the resulting date is as least as large as the minDate.

Usage

```
obfuscateDate(baseDate, minDate, maxDelta = 30L)
```

Arguments

baseDate	vector of Date values with dates to be obfuscated
minDate	optional vector of Date values, the same length as baseDate, that has the lower bound of resulting obfuscated dates. When missing, the lower bound is each baseDate minus its maxDelta, so the bound never binds. An NA in minDate leaves that element without a lower bound. A minDate on or after baseDate plus maxDelta (a date no draw can reach) stops with an error that names the element.
maxDelta	integer vector of length 1 or the same length as baseDate that is used to create min and max arguments to runif (runif(n, min = 0, max = 1))

Value

A vector of dates that have be obfuscated.

See Also

Other obfuscation: mapIdsToObfuscated(), obfuscateGenomicROH(), obfuscateGenotypeMatrix(), obfuscateId(), obfuscateLdBlocks(), obfuscateMhcHaplotypes(), obfuscatePed(), obfuscateTwinRelations()

Examples

```
library(nprcgenekeepr)
someDates <- rep(
  as.Date(c("2009-2-16", "2016-2-16"), format = "%Y-%m-%d"),
  10
)
minBirthDate <- rep(as.Date("2009-2-16", format = "%Y-%m-%d"), 20)
obfuscateDate(someDates, minBirthDate, 30L)
```

obfuscateGenomicROH *De-identify a computeGenomicROH() result table*

Description

Remaps the id column of a computeGenomicROH result table through the same alias vector obfuscatePed(..., map = TRUE) already returns, mirroring obfuscateTwinRelations's and obfuscateGenotypeMatrix's pattern (issue #152 Slice 5, design decision D7: "any sequence-derived export... routes through a new de-identification primitive"). computeGenomicROH's output is a plain per-individual data.frame keyed by a single id column – a different shape from every prior obfuscate*() sibling, so none of them fit directly.

Usage

```
obfuscateGenomicROH(rohTable, map)
```

Arguments

rohTable	data.frame as returned by computeGenomicROH: at least an id column.
map	named character vector of aliases, keyed by the original id – the map element of obfuscatePed(..., map = TRUE)'s return value.

Details

A row whose id is absent from map stop()s rather than silently dropping or leaking the real id.

Value

rohTable with id replaced by its alias; every other column (nSegments, totalRohLength, fRoh) is unchanged.

See Also

Other obfuscation: mapIdsToObfuscated(), obfuscateDate(), obfuscateGenotypeMatrix(), obfuscateId(), obfuscateLdBlocks(), obfuscateMhcHaplotypes(), obfuscatePed(), obfuscateTwinRelations()

Examples

```
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A01", "A02"),
  sire = c(NA, NA),
  dam = c(NA, NA),
  sex = c("M", "F"),
  stringsAsFactors = FALSE
)
rohTable <- data.frame(
  id = c("A01", "A02"), nSegments = c(1L, 0L),
```

```
  totalRohLength = c(1000000, 0), fRoh = c(0.1, 0),
  stringsAsFactors = FALSE
)
obfuscated <- obfuscatePed(ped, map = TRUE)
obfuscateGenomicROH(rohTable, obfuscated$map)
```

`obfuscateGenotypeMatrix` *De-identify a sequence-scale genotype matrix*

Description

Remaps the row names (individual `id`s) of a `buildMarkerGenotypeMatrix`-shaped wide genotype matrix through the same alias vector `obfuscatePed(..., map = TRUE)` already returns, mirroring `obfuscateTwinRelations`'s and `obfuscateLdBlocks`'s pattern. Genotype/allele values themselves are never perturbed – unlike a date, there is no scientifically-valid "obfuscation" of an allele call that preserves validity while hiding identity, so the only real protection this primitive provides is which people see the exported file at all (issue #152 Slice 4, design decision D7).

Usage

```
obfuscateGenotypeMatrix(genotypeMatrix, map)
```

Arguments

`genotypeMatrix`	a character matrix as returned by `buildMarkerGenotypeMatrix`: rows are individual `id`s, columns are loci.
`map`	named character vector of aliases, keyed by the original id – the `map` element of `obfuscatePed(..., map = TRUE)`'s return value.

Details

A row whose id is absent from `map` `stop()`s rather than silently dropping or leaking the real id.

Value

`genotypeMatrix` with row names replaced by their aliases; column names and every genotype cell value are unchanged.

See Also

Other obfuscation: `mapIdsToObfuscated()`, `obfuscateDate()`, `obfuscateGenomicROH()`, `obfuscateId()`, `obfuscateLdBlocks()`, `obfuscateMhcHaplotypes()`, `obfuscatePed()`, `obfuscateTwinRelations()`

Examples

```
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A01", "A02"),
  sire = c(NA, NA),
  dam = c(NA, NA),
  sex = c("M", "F"),
  stringsAsFactors = FALSE
)
genotypeMatrix <- matrix(
  c("A/A", "A/B"), nrow = 2L, dimnames = list(c("A01", "A02"), "L1")
)
obfuscated <- obfuscatePed(ped, map = TRUE)
obfuscateGenotypeMatrix(genotypeMatrix, obfuscated$map)
```

---

obfuscateId *Create ID aliases of a specified length*

Description

ID aliases are pseudorandom sequences of alphanumeric upper case characters where the letter "O" is not included for readability.. User has the option of providing a character vector of aliases to avoid using. Because aliases are alphanumeric, they never contain a period ("."), honoring the ID rule enforced at data input by qcStudbook.

Usage

```
obfuscateId(id, size = 10L, existingIds = character(0L), placeholder = NULL)
```

Arguments

id	character vector of IDs to be obfuscated (alias creation). It must not contain NA; an NA id stops with a message that gives its position.
size	character length of each alias. An ID that is a placeholder for an unknown parent (see removeAutoGenIds) gets an alias of the same shape, the placeholder prefix plus random characters; when size is too short for that shape to be recognized (shorter than 5 for the default "U%04d" format), only those aliases are made longer, to the shortest recognizable length.
existingIds	character vector of existing aliases to avoid duplication.
placeholder	optional logical vector, one value per id, saying which ids are placeholders for unknown parents (TRUE) and which are real animals (FALSE), as recorded in the placeholder column that qcStudbook writes. A placeholder gets a placeholder-shaped alias and a real animal a real-shaped one, even when the id's own shape says otherwise (a real animal named "U1234"). NA or NULL (the default) reads the id by its shape.

Value

A named character vector of aliases where the name is the original ID value.

See Also

Other obfuscation: `mapIdsToObfuscated()`, `obfuscateDate()`, `obfuscateGenomicROH()`, `obfuscateGenotypeMatrix()` `obfuscateLdBlocks()`, `obfuscateMhcHaplotypes()`, `obfuscatePed()`, `obfuscateTwinRelations()`

Examples

```
library(nprcgenekeepr)
integerIds <- 1L:10L
obfuscateId(integerIds, size = 4L)
characterIds <- paste0(paste0(sample(LETTERS, 1L, replace = FALSE)), 1L:10L)
obfuscateId(characterIds, size = 4L)
```

---

obfuscateLdBlocks *De-identify a markerLdBlock() result table*

Description

Remaps the `idsUsed` column of a `markerLdBlock` result table through the same alias vector `obfuscatePed(..., map = TRUE)` already returns, mirroring `obfuscateTwinRelations`'s pattern. `markerLdBlock`'s output is otherwise a locus-pair-level population statistic table with no per-individual ids – `idsUsed` (populated only when `markerLdBlock` was called with `founderIds`) is the only place a real id can appear, and any exported block/LD statistic table must route through the same curator-controlled de-identification gate issue #150 established (D9): a joint, multi-locus statistic carries *more* identifying power than a single-locus one, not less.

Usage

```
obfuscateLdBlocks(ldBlockResult, map)
```

Arguments

`ldBlockResult`	data.frame as returned by `markerLdBlock`: at least an `idsUsed` column (comma-joined ids, or `NA`).
`map`	named character vector of aliases, keyed by the original id – the `map` element of `obfuscatePed(..., map = TRUE)`'s return value.

Details

A row whose `idsUsed` contains an id absent from `map` `stop()`s rather than silently dropping or leaking the real id – `markerLdBlock`'s own `founderIds` contract should already guarantee every id is a valid pedigree id, so this is a defensive check, not the primary validation path.

Value

ldBlockResult with idsUsed ids replaced by their aliases (comma-joined, same order); a row whose idsUsed is NA is returned unchanged. Every other column is unchanged.

See Also

Other obfuscation: mapIdsToObfuscated(), obfuscateDate(), obfuscateGenomicROH(), obfuscateGenotypeMatrix() obfuscateId(), obfuscateMhcHaplotypes(), obfuscatePed(), obfuscateTwinRelations()

Examples

```
library(nprcgenekeepr)
ped <- data.frame(
  id = c("F1", "F2", "S1", "S2"),
  sire = c(NA, NA, "F1", "F1"),
  dam = c(NA, NA, "F2", "F2"),
  sex = c("M", "F", "F", "F"),
  stringsAsFactors = FALSE
)
ldBlockResult <- data.frame(
  locus1 = "L1", locus2 = "L2", chrom = "1", Dprime = 0.5, r2 = 0.3,
  nUsed = 2L, idsUsed = "F1,F2", caveat = "Descriptive statistic only.",
  stringsAsFactors = FALSE
)
obfuscated <- obfuscatePed(ped, map = TRUE)
obfuscateLdBlocks(ldBlockResult, obfuscated$map)
```

---

obfuscateMhcHaplotypes

De-identify an MHC haplotype carrier table

Description

Remaps the id column of a mhcHaplotypeCarriers table through the same alias vector obfuscatePed(..., map = TRUE) already returns. obfuscatePed scrubs exactly one pedigree data frame and cannot reach a second, sidecar object – this is the companion scrub an MHC carrier table needs so an "obfuscated" export never leaks real animal ids while the main pedigree is de-identified.

Usage

```
obfuscateMhcHaplotypes(carriers, map)
```

Arguments

carriers	data.frame with columns haplotype, id, uncertain as returned by mhcHaplotypeCarriers.
map	named character vector of aliases, keyed by the original id – the map element of obfuscatePed(..., map = TRUE)'s return value.

Details

Haplotype labels are left byte-identical: an MHC haplotype name is a shared nomenclature term, not an animal identifier, and there is no validity-preserving way to obfuscate one. Only `id` is ever remapped – a map entry whose name happens to match a haplotype label never touches the `haplotype` column. The `uncertain` disclosure column passes through unchanged.

A row whose `id` is absent from `map` `stop()`s rather than silently dropping or leaking the real id.

Value

`carriers` with `id` replaced by its alias; `haplotype` and `uncertain` are unchanged.

See Also

`mhcHaplotypeCarriers`, `obfuscatePed`

Other obfuscation: `mapIdsToObfuscated()`, `obfuscateDate()`, `obfuscateGenomicROH()`, `obfuscateGenotypeMatrix()` `obfuscateId()`, `obfuscateLdBlocks()`, `obfuscatePed()`, `obfuscateTwinRelations()`

Examples

```
ped <- data.frame(
  id = c("F1", "F2", "S1", "S2"),
  sire = c(NA, NA, "F1", "F1"),
  dam = c(NA, NA, "F2", "F2"),
  sex = c("M", "F", "F", "F"),
  stringsAsFactors = FALSE
)
genotype <- data.frame(
  id = c("S1", "S2"),
  haplotype1 = c("A001_B001", "A001_B001"),
  haplotype2 = c("A002_B012", "A008_B015b"),
  stringsAsFactors = FALSE
)
carriers <- mhcHaplotypeCarriers(genotype, rareOnly = FALSE)
obfuscated <- obfuscatePed(ped, map = TRUE)
obfuscateMhcHaplotypes(carriers, obfuscated$map)
```

---

obfuscatePed *Obfuscate a pedigree by aliasing IDs and shifting dates*

Description

User provides a pedigree object (`ped`), the number of characters to be used for alias IDs (`size`), and the maximum number of days that the birthdate can be shifted (`maxDelta`).

Usage

```
obfuscatePed(
  ped,
  size = 6L,
  maxDelta = 30L,
  existingIds = character(0L),
  map = FALSE,
  linkedDateShift = TRUE
)
```

Arguments

ped	The pedigree information in data.frame format
size	integer value indicating number of characters in alias IDs. IDs that stand for unknown parents get longer aliases when size is too short to keep them recognizable as placeholders (see obfuscateId).
maxDelta	integer value indicating maximum number of days that the birthdate can be shifted
existingIds	character vector of existing aliases to avoid duplication.
map	logical if TRUE a list object is returned with the new pedigree and a named character vector with the names being the original IDs and the values being the new alias values. Defaults to FALSE.
linkedDateShift	logical, defaults to TRUE. When TRUE, every Date column of one individual is shifted by the same, single random offset (drawn once per individual), so the gaps between an individual's own dates (e.g. birth, exit, death) are preserved exactly – avoiding a defect where independently-shifted date columns can invert an individual's recorded date order (e.g. an obfuscated exit preceding an obfuscated birth), producing a negative recomputed age (issue #150 D3). When FALSE, each Date column is shifted independently via obfuscateDate (the original, pre-#150 behavior).

Details

Any name column is overwritten with NA. Every Date column (e.g. birth, exit, death) is shifted, not only the birthdate. When the age, birth and exit columns are all present and birth is a Date, age is recomputed from the shifted dates.

Value

An obfuscated pedigree: IDs aliased, name set to NA, Date columns shifted and age recomputed. With map = TRUE, a list holding that pedigree and the alias map.

See Also

Other obfuscation: mapIdsToObfuscated(), obfuscateDate(), obfuscateGenomicROH(), obfuscateGenotypeMatrix() obfuscateId(), obfuscateLdBlocks(), obfuscateMhcHaplotypes(), obfuscateTwinRelations()

Examples

```
library(nprcgenekeepr)
ped <- qcStudbook(nprcgenekeepr::pedGood)
obfuscatedPed <- obfuscatePed(ped)
ped
obfuscatedPed
```

`obfuscateTwinRelations` *De-identify a twin/zygosity relations table*

Description

Remaps `id1`/`id2` in a twin/zygosity sidecar table (see `checkTwinRelations`) through the same alias vector `obfuscatePed(..., map = TRUE)` already returns. `obfuscatePed` scrubs exactly one pedigree data frame and cannot reach a second, sidecar object – this is the companion scrub a twin-relations table needs so an "obfuscated" export never leaks real ids through the sidecar table while the main pedigree is de-identified.

Usage

```
obfuscateTwinRelations(twinRelations, map)
```

Arguments

`twinRelations`	data.frame with columns `id1`, `id2`, `code`. See `checkTwinRelations`.
`map`	named character vector of aliases, keyed by the original id – the `map` element of `obfuscatePed(..., map = TRUE)`'s return value.

Details

A row whose `id1` or `id2` is absent from `map` `stop()`s rather than silently dropping or leaking the real id. `checkTwinRelations`'s own existence rule should already have excluded that case upstream against the un-obfuscated pedigree, so this is a defensive check, not the primary validation path.

Value

`twinRelations` with `id1`/`id2` replaced by their aliases; `code` is unchanged.

See Also

Other obfuscation: `mapIdsToObfuscated()`, `obfuscateDate()`, `obfuscateGenomicROH()`, `obfuscateGenotypeMatrix()`, `obfuscateId()`, `obfuscateLdBlocks()`, `obfuscateMhcHaplotypes()`, `obfuscatePed()`

Examples

```
ped <- data.frame(
  id = c("F1", "F2", "S1", "S2"),
  sire = c(NA, NA, "F1", "F1"),
  dam = c(NA, NA, "F2", "F2"),
  sex = c("M", "F", "F", "F"),
  stringsAsFactors = FALSE
)
twinRelations <- data.frame(
  id1 = "S1", id2 = "S2", code = "MZ twin", stringsAsFactors = FALSE
)
obfuscated <- obfuscatePed(ped, map = TRUE)
obfuscateTwinRelations(twinRelations, obfuscated$map)
```

---

offspringCounts *Tabulate offspring counts, optionally by population*

Description

Optionally find the number that are part of the population of interest.

Usage

```
offspringCounts(probands, ped, considerPop = FALSE)
```

Arguments

probands	character vector of egos for which offspring should be counted.
ped	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam; population is also read when considerPop is TRUE). This is the complete pedigree. Ids given as numbers or as a factor in sire and dam are taken as their text.
considerPop	logical value indication whether or not the number of offspring that are part of the focal population are to be counted? Default is FALSE. If ped has no population column, considerPop has no effect and only totalOffspring is returned.

Value

A dataframe containing the column totalOffspring (and livingOffspring when considerPop is TRUE and ped has a population column). The animal ids are the data frame row names when probands are unique; duplicated probands give sequential row names.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2,
  minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
counts <- offspringCounts(probands, ped)
```

---

ped1Alleles *Gene-drop alleles example (baboon pedigree)*

Description

A dataframe created by the geneDrop function.

Usage

```
data(ped1Alleles)
```

Format

A dataframe with 554 rows and 6 variables

V1 alleles assigned to the parents of the animals identified in the id column during iteration 1 of gene dropping performed by geneDrop.

V2 alleles assigned to the parents of the animals identified in the id column during iteration 2 of gene dropping performed by geneDrop.

V3 alleles assigned to the parents of the animals identified in the id column during iteration 3 of gene dropping performed by geneDrop.

V4 alleles assigned to the parents of the animals identified in the id column during iteration 4 of gene dropping performed by geneDrop.

id character vector of animal IDs provided to the gene dropping function geneDrop.

parent the parent type ("sire" or "dam") of the parent who supplied the alleles as assigned during each of the 4 gene dropping iterations performed by geneDrop.

Source

example baboon pedigree file provided by Deborah Newman, Southwest National Primate Center.

pedDuplicateIds *Example studbook with a duplicated record*

Description

A data frame with 9 rows and 5 columns (ego_id, sire.id, dam_id, sex, birth_date) representing a full pedigree with a duplicated record.

Usage

```
data(pedDuplicateIds)
```

Format

An object of class `data.frame` with 9 rows and 5 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

pedFemaleSireMaleDam *Example studbook with sex-mismatched parents*

Description

A data frame with 8 rows and 5 columns (ego_id, sire.id, dam_id, sex, birth_date) representing a full pedigree with the errors of having a sire labeled as female and a dam labeled as male.

Usage

```
data(pedFemaleSireMaleDam)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

pedGood *Valid example studbook (no QC errors)*

Description

A data frame with 8 rows and 5 columns (ego_id, sire.id, dam_id, sex, birth_date) representing a full pedigree with no errors.

Usage

```
data(pedGood)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

pedInvalidDates *Example studbook with invalid birth dates*

Description

A data frame with 8 rows and 5 columns (id, sire, dam, sex, birth) representing a full pedigree with values in the `birth` column that are not valid dates.

Usage

```
data(pedInvalidDates)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

pedMissingBirth *Example studbook missing the birth date column*

Description

A data frame with 8 rows and 4 columns (ego_id, sire.id, dam_id, sex) representing a full pedigree that is missing the birth_date column.

Usage

```
data(pedMissingBirth)
```

Format

An object of class `data.frame` with 8 rows and 4 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

pedOne *Raw pedigree-file fragment for testing (5 columns)*

Description

A loadable version of a pedigree file fragment used for testing and demonstration.

Usage

```
data(pedOne)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Examples

```
library(nprcgenekeepr)
data("pedOne")
head(pedOne)
```

pedSameMaleIsSireAndDam *Example studbook with a male as both sire and dam*

Description

A data frame with 8 rows and 5 columns (ego_id, sire.id, dam_id, sex, birth_date) representing a full pedigree in which the same male animal is listed as both a sire and a dam.

Usage

```
data(pedSameMaleIsSireAndDam)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

pedSix *Raw pedigree-file fragment for testing (7 columns)*

Description

A loadable version of a pedigree file fragment used for testing and demonstration.

Usage

```
data(pedSix)
```

Format

An object of class `data.frame` with 8 rows and 7 columns.

Examples

```
library(nprcgenekeepr)
data("pedSix")
head(pedSix)
```

pedWithGenotype *Pedigree with simulated genotypes (from qcPed)*

Description

A dataframe produced from qcPed by adding made up genotypes.

Usage

```
data(pedWithGenotype)
```

Format

A dataframe containing 280 records with 12 columns: `id`, `sire`, `dam`, `sex` (factor), `gen`, `birth`, `exit`, `age`, `first` and `second` (numeric allele codes), and `first_name` and `second_name` (the allele names).

pedWithGenotypeReport *Genetic-value report for pedWithGenotype*

Description

A list containing the output of `reportGV`.

Usage

```
data(pedWithGenotypeReport)
```

Format

An object of class `list` (inherits from `nprcgenekeeprGV`) of length 14.

Source

pedWithGenotypeReport was made with pedWithGenotype as input into reportGV with 10,000 iterations.

pedWithGenotypeReport is a simple example report for use in examples and unit tests. It was created using the following commands.

- set.seed(10)
- pedWithGenotypeReport <- reportGV(nprcgenekeepr::pedWithGenotype, guIter = 10000)
- save(pedWithGenotypeReport, file = "data/pedWithGenotypeReport.RData")

Examples

```
pedWithGenotypeReport <- nprcgenekeepr::pedWithGenotypeReport
```

`plotSnapshotTrends` *Plot colony genetic-health trends from a snapshot history*

Description

Draws the longitudinal trend view of a colony snapshot history (issue #167): one `ggplot` object faceted per metric (free y scales), with `snapshotDate` on the x axis and one colored series per `membershipRule`, so successive snapshots are compared like with like. Sampling-uncertainty ribbons are drawn exactly where the schema carries a standard error: `fg` (`fgSE`) and `meanGu` (`meanGuSE`) — the Monte Carlo gene-drop uncertainty is surfaced, never hidden. Date labels that would overlap on a narrow panel are dropped instead of drawn over one another.

Usage

```
plotSnapshotTrends(history, metrics = NULL, ncol = NULL)
```

Arguments

`history`	data.frame holding the snapshot history in the 27-column version-1 schema; validated internally with `checkSnapshotHistory`. A trend needs at least two snapshots.
`metrics`	character vector naming the metric columns to facet. The default `NULL` plots the 18 metric columns (the `reportGV()` colony scalars and the Summary Statistics aggregates); the composition counts (`nAnimals`, `nMales`, `nFemales`) may be requested to make membership churn visible.
`ncol`	single positive whole number: how many columns of panels to draw. The default `NULL` leaves the grid to `ggplot2` (5 columns for the 18 default metrics). A narrow figure needs fewer columns, so each panel stays wide enough for its title; the Genetic-Health Trends tab picks the number from the width of its plot. A number above the count of panels draws one column for each panel.

Details

Snapshots whose provenance (`guIter`, `guThresh`, or `packageVersion`) changed relative to the same rule's previous snapshot are drawn with a distinct point shape, and the plot carries a caption naming the changed fields: mixed-provenance series are flagged, not refused.

Snapshots rebuilt from studbook dates by `backfillColonySnapshots` are filed under the rule `"reconstructedLiving"` (issue #170). That series is drawn as a dashed line, the recorded rules stay solid, and the caption says the dashed lines are rebuilt and were not recorded at the time.

Value

A `ggplot` object.

Examples

```
history <- checkSnapshotHistory(readSnapshotHistory(system.file("extdata",
  "examples", "example_snapshot_history.csv",
  package = "nprcgenekeepr"
)))
p <- plotSnapshotTrends(history, metrics = c("fe", "fg", "meanGu"))
```

`previewBackfillSupport` *Count what each date of a rebuilt series rests on, before building it*

Description

Returns the `support` table that `backfillColonySnapshots` reports for the same dates, without the gene drop, so the counts are there in a moment and a start year can be chosen knowingly before minutes of analysis are spent (issue #170). The Genetic-Health Trends tab shows this same table before its "Build earlier snapshots" button is used.

Usage

```
previewBackfillSupport(
  ped,
  dates = NULL,
  from = NULL,
  to = NULL,
  by = "year",
  minAnimals = 2L
)
```

Arguments

`ped`	A quality-controlled pedigree data.frame (see `qcStudbook`) holding at least `id`, `sire`, `dam`, `sex`, `gen`, `birth` and `exit` (the last two as `Date` columns), and a `status` column when the studbook has one. Pass the full pedigree, not one already trimmed to a focal list of animals.
`dates`	The dates to count: `Date` values or ISO-8601 (YYYY-MM-DD) strings, at least one, with no missing or duplicated value. Leave it out (`NULL`) to use the series described by `from`, `to` and `by`; it cannot be combined with them.
`from`	The first date of the series: a `Date` or an ISO-8601 string. The default is the first 1 January with at least `minAnimals` living animals.
`to`	The last date of the series, as `from`. The default is the last 1 January on or before the latest birth or exit date in `ped`.
`by`	The step of the series: a string for `seq.Date`, such as `"year"` (the default), `"6 months"` or `"quarter"`.
`minAnimals`	Single whole number, at least 2: the fewest living animals a 1 January needs to start the default series. Default 2.

Details

The dates are laid out exactly as `backfillColonySnapshots` lays them out: give `dates`, or leave it out for a series of `by` steps (one year by default) from `from` to `to`. Unless you give them, the series starts at the first 1 January with at least `minAnimals` living animals and ends at the last 1 January on or before the latest birth or exit date in the pedigree. `minAnimals` only moves that default start: a date you name, in `dates` or as `from`, is never refused for having few animals.

An animal is counted on a date by the rules of `backfillColonySnapshots`: alive from its birth date, or, with no birth date, from the birth of its first offspring that has a date, until its exit date; an animal with no exit date counts only when the studbook lists it as living. No threshold refuses a date; read the counts beside the trend.

Value

A data.frame with one row for each date, in date order, and the columns of the `support` table of `backfillColonySnapshots`: `snapshotDate`, `nLiving` (the animals counted), `nPlacedByOffspring` (of those, the ones with no birth date), `nPossiblyAlive` (animals with no birth date that are not counted and had not left), `pctSireKnown` and `pctDamKnown` (the percentage of the living animals whose sire or dam is recorded, that is neither blank nor a generated stand-in; `NA` when nobody was alive).

See Also

`backfillColonySnapshots`

Examples

```
ped <- nprcgenekeepr::qcPed
previewBackfillSupport(ped,
  dates = as.Date(c("2005-01-01", "2010-01-01"))
)

## the default series, and a later start when the early years rest on little
head(previewBackfillSupport(ped), 3L)
head(previewBackfillSupport(ped, minAnimals = 30L), 3L)
```

`print.summary.nprcgenekeeprErr`

Print an nprcgenekeepr summary object

Description

Print an nprcgenekeepr summary object

Usage

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

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

Arguments

x	object of class summary.nprcgenekeeprErr (from `summary()` of a studbook error list) or summary.nprcgenekeeprGV (from `summary()` of a genetic value report)
...	further arguments passed to the `print()` call for the suspicious-parents table (and ignored by the GV method)

Value

The summary object, returned invisibly after it is printed.

The summary object, returned invisibly after it is printed.

Examples

```
library(nprcgenekeepr)
errorLst <- qcStudbook(nprcgenekeepr::pedInvalidDates,
  reportChanges = TRUE, reportErrors = TRUE
)
summary(errorLst)
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedGood
ped <- suppressWarnings(qcStudbook(ped, reportErrors = FALSE))
summary(reportGV(ped, guIter = 10))
```

`processQcStudbookResult` *Process qcStudbook Result into UI-Friendly Format*

Description

Converts the errorLst object returned by qcStudbook (when reportErrors=TRUE) into a format suitable for display in the Shiny UI.

Usage

```
processQcStudbookResult(errorLst)
```

Arguments

errorLst	list object returned by qcStudbook with reportErrors = TRUE. A NULL is treated as a failed check (hasErrors is TRUE with the error "No result returned from quality control check"), yet qcStudbook returns NULL when it finds no errors and no column changes. Use runQcStudbook, which handles that case, or pass getEmptyErrorLst() instead of NULL. Expected to be of class nprcgenekeeprErr containing error fields such as femaleSires, maleDams, sireAndDam, duplicateIds, invalidIdChars, invalidPlaceholderRows, missingColumns, invalidDateRows, suspiciousParents, failedDatabaseConnection, and changedCols.

Value

A list with the following components:

- errors - data.frame with columns Row, Error, Details
- warnings - data.frame with columns Row, Warning, Details
- changedCols - list of changed column information
- hasErrors - logical indicating if any errors were found
- hasChangedCols - logical indicating if columns were renamed

See Also

qcStudbook for generating the errorLst input

runQcStudbook for a wrapper that uses this function

checkErrorLst for checking if errorLst has errors

checkChangedColsLst for checking column changes

Examples

```
library(nprcgenekeepr)
## Turn a qcStudbook error list into UI-friendly data frames.
errorLst <- qcStudbook(nprcgenekeepr::pedFemaleSireMaleDam,
  reportErrors = TRUE
)
result <- processQcStudbookResult(errorLst)
result$hasErrors
result$errors
```

---

qcBreeders *Potential breeder IDs (29 baboons)*

Description

A character vector of 29 baboon IDs that are potential breeders.

Usage

```
data(qcBreeders)
```

Format

An object of class `character` of length 29.

Source

qcBreeders is a character vector of 3 males and 26 females from the `qcPed` data set.

These 29 animal IDs are used for examples and unit tests. They were initially selected for having low kinship coefficients.

---

qcPed *Example quality-controlled baboon pedigree*

Description

A data frame with 280 rows and 8 columns.

id character column of animal IDs

sire the male parent of the animal indicated by the `id` column.

dam the female parent of the animal indicated by the `id` column.

sex sex of the animal indicated by the `id` column.

gen generation number (numeric, whole numbers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth birth date in `Date` format of the animal indicated by the `id` column.

exit exit date in `Date` format of the animal indicated by the `id` column.

age age in years (numeric) of the animal indicated by the `id` column.

Usage

```
data(qcPed)
```

Format

An object of class `data.frame` with 280 rows and 8 columns.

qcPedGvReport *Genetic-value report for qcPed*

Description

qcPedGvReport is a genetic value report for illustrative purposes only. It is used in examples and unit tests with the nprcgenekeepr package. It was created using the following commands.

- set.seed(10)
- qcPedGvReport <- reportGV(nprcgenekeepr::qcPed, guIter = 10000)
- save(qcPedGvReport, file = "data/qcPedGvReport.RData")

Usage

```
data(qcPedGvReport)
```

Format

An object of class `list` (inherits from `nprcgenekeeprGV`) of length 14.

Examples

```
qcPedGvReport <- nprcgenekeepr::qcPedGvReport
```

qcStudbook *Run quality control on a studbook or pedigree*

Description

Main pedigree curation function that performs basic quality control on pedigree information

Usage

```
qcStudbook(
  sb,
  minSireAge = NULL,
  minDamAge = NULL,
  minParentAge = lifecycle::deprecated(),
  reportChanges = FALSE,
  reportErrors = FALSE
)
```

Arguments

sb	A dataframe containing a table of pedigree and demographic information. The function recognizes the following columns (optional columns will be used if present, but are not required): • id — Character vector with Unique identifier for all individuals • sire — Character vector with unique identifier for the father of the current id • dam — Character vector with unique identifier for the mother of the current id • sex — Factor (levels: "M", "F", "U") Sex specifier for an individual • birth — Date or NA (optional) with the individual's birth date • departure — Date or NA (optional) an individual was sold or shipped from the colony • death — date or NA (optional) Date of death, if applicable • status — Factor (levels: ALIVE, DECEASED, SHIPPED, UNKNOWN) (optional) Status of an individual (see convertStatusCodes) • origin — Character or NA (optional) Facility an individual originated from, if other than ONPRC • ancestry — Character or NA (optional) Geographic population to which the individual belongs • spf — Character or NA (optional) Specific pathogen-free status of an individual • vasxOvx — Character or NA (optional) Indicator of the vasectomy/ovariectomy status of an animal; NA if animal is intact, assume all other values indicate surgical alteration • condition — Character or NA (optional) Indicator of the restricted status of an animal. "Nonrestricted" animals are generally assumed to be naive. • placeholder — Logical or NA (optional) TRUE marks a made-up stand-in for an unknown parent, FALSE a real animal. TRUE, true, True, T and 1 count as TRUE; FALSE, false, False, F and 0 as FALSE; a blank as unmarked. Any other value is an error (listed in errorLst$invalidPlaceholderRows with reportErrors == TRUE, otherwise the function throws an error naming the rows).
minSireAge	numeric minimum age in years for a male to have sired an offspring. NULL (default) looks up the floor for each sire's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor.
minDamAge	numeric minimum age in years for a female to have borne an offspring. NULL (default) looks up the floor for each dam's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor.
minParentAge	**[Deprecated]** Deprecated scalar minimum parent age. Supplying it sets both minSireAge and minDamAge; use those sex-specific parameters instead.
reportChanges	logical value that if TRUE, the errorLst contains the list of changes made to the column names. Default is FALSE.

reportErrors logical value if TRUE will scan the entire file and report back changes made to input and errors in a list of list where each sublist is a type of change or error found. Changes will include column names, case of categorical values (male, female, unknown), etc. Errors will include missing columns, invalid date rows, male dams, female sires, an ID that is both a sire and a dam, duplicated IDs, IDs with invalid characters, invalid placeholder values, and records with one or more parents below minimum age of parents.

The following changes are made to the column names.

- Column names are converted to all lower case
- Spaces, periods (".") and underscores ("_") within column names are removed
- egoid and ego are converted to id
- sireid is converted to sire
- damid is converted to dam
- birthdate and deathdate are converted to birth and death
- recordstatus, fromcenter and geographicorigin are converted to recordStatus, fromCenter and geographicOrigin

If the dataframe (sb) does not contain the five required columns (id, sire, dam, sex, and birth) and reportErrors == FALSE, the function throws an error by calling stop(). With reportErrors == TRUE the missing names are returned in errorLst$missingColumns.

Animal IDs (id, sire, dam) must not contain a period ("."); other characters are accepted. Periods cause problems across software environments (R column-name and formula parsing, file-name extensions, programming-language namespaces, and regular expressions), so any id, sire, or dam value containing a period is treated as an error. With reportErrors == TRUE the offending values are returned in errorLst$invalidIdChars; otherwise the function throws an error. All automatically generated IDs (see addUIds) honor this rule.

If the id field has the string *UNKNOWN* (any case), the record is removed. A record whose sire and dam are both NA or *UNKNOWN* (any case) is kept as a founder; *UNKNOWN* in either field becomes NA. If only one of the fields sire or dam is missing, the missing parent is replaced with a unique identifier with the form Unnnn, where nnnn represents one of a series of sequential integers representing the number of missing sires and dams right justified in a pattern of 0000. See addUIds function.

The function addParents is used to add records for parents missing their own record in the pedigree.

The function convertSexCodes is used with ignoreHerm == TRUE to convert sex codes according to the following factors of standardized codes:

- F – replacing "FEMALE" or "2"
- M – replacing "MALE" or "1"
- H – replacing "HERMAPHRODITE" or "4", if ignoreHerm == FALSE
- U – replacing "HERMAPHRODITE" or "4", if ignoreHerm == TRUE
- U – replacing "UNKNOWN" or "3"
- U – replacing a missing, blank or unrecognized value

Case and any spaces around a code are ignored. qcStudbook always uses the default ignoreHerm == TRUE, so H is never assigned, although the returned factor keeps the level H.

The function correctParentSex is used to ensure no parent is both a sire and a dam. If this error is detected and reportErrors == FALSE, the function throws an error and halts the program (with reportErrors == TRUE the IDs are returned in errorLst$sireAndDam). It also recodes a female sire as male and a male dam as female; with reportErrors == FALSE this is silent, otherwise the IDs are reported in errorLst$femaleSires and errorLst$maleDams.

The function convertStatusCodes converts status indicators to the following factors of standardized codes. Case of the original status value is ignored.

- "ALIVE" — replacing "alive", "A" and "1"
- "DECEASED" — replacing "deceased", "DEAD", "D", "2"
- "SHIPPED" — replacing "shipped", "sold", "sale", "s", "3"
- "UNKNOWN" — replacing is.na(status)
- "UNKNOWN" — replacing "unknown", "U", "4"

The function convertAncestry converts ancestry indicators using regular expressions such that the following conversions are made from character strings that match selected substrings to the following factors.

- "INDIAN" — replacing "ind" and not "chin"
- "CHINESE" — replacing "chin" and not "ind"
- "HYBRID" — replacing "hyb" or "chin" and "ind"
- "JAPANESE" — replacing "jap"
- "UNKNOWN" — replacing NA
- "OTHER" — replacing not matching any of the above

The function convertDate converts character representations of dates in the columns birth, death, departure, and exit to dates using the as.Date function.

The function setExit uses heuristics and the columns death and departure to set exit if it is not already defined.

The function calcAge uses the birth and the exit columns to define the age column. The numerical value is rounded to the nearest 0.1 of a year. If exit is not defined, the current system date (Sys.Date()) is used.

The function findGeneration is used to define the generation number for each animal in the pedigree.

The function removeDuplicates checks for any duplicated records and removes the duplicates. It also throws an error and stops the program if an ID appears in more than one record where one or more of the other columns have a difference.

The standard columns come first, followed by any unrecognized columns, which are retained. The rows are ordered by generation number and then ID.

Finally the columns id, sire, and dam are coerced to character.

The returned pedigree always has a logical placeholder column with no NA: TRUE for the stand-ins made here (see addUIds), the value the input gave for any

other row that had one, and for the rest TRUE when the ID looks like a stand-in (see removeAutoGenIds). A center can mark a real animal whose ID looks like a stand-in (for example "U1234") as FALSE.

Value

With reportErrors == FALSE, a data.frame with standardized and quality controlled pedigree information. With reportErrors == TRUE, an nprcgenekeeprErr list of the changes and errors found, or NULL when there are no errors and no column changes to report.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
ped <- qcStudbook(examplePedigree,
  minSireAge = 2.0, minDamAge = 2.0, reportChanges = FALSE,
  reportErrors = FALSE
)
names(ped)
```

---

rankSubjects *Rank animals by genetic value*

Description

Part of Genetic Value Analysis

Usage

```
rankSubjects(rpt)
```

Arguments

rpt	a named list of data.frames containing genetic value data for the population, as made by the report ordering step. The element names decide the designation: lowVal and noParentage as described above, and every other element (for example imports, lowMk and highGu) is "High Value". The tiers separate out the animals that are imports, those with low mean kinship (a mean-kinship z-score at or below zScoreCutoff), those with high genome uniqueness (gu above guCutoff), and the remainder; see reportGV.

Details

Adds a rank column to each data.frame in rpt: integers from 1 to the total number of ranked animals, running on from one tier to the next in the order of the list. Animals in the noParentage element get an NA rank. Adds a value column designating each animal "High Value", "Low Value" (the lowVal element) or "Undetermined" (the noParentage element).

Value

A list of dataframes with value and ranking information added. Elements with no rows are returned unchanged.

References

Vinson, A. and Raboin, M.J. (2015) "A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques (*Macaca mulatta*)" *Journal of the American Association for Laboratory Animal Science*, 2015 Nov, Vol.54(6), pp.700-707.

Examples

```
library(nprcgenekeepr)
finalRpt <- nprcgenekeepr::finalRpt
rpt <- rankSubjects(nprcgenekeepr::finalRpt)
rpt[["highGu"]][1, "value"]
rpt[["highGu"]][1, "rank"]
rpt[["lowMk"]][1, "value"]
rpt[["lowMk"]][1, "rank"]
rpt[["lowVal"]][1, "value"]
rpt[["lowVal"]][1, "rank"]
```

---

readAncestryRules *Read an ancestry compatibility rules table from a file*

Description

Reads a center-configurable ancestry compatibility rules table from a user-supplied file into a data frame for `checkAncestryRules` (issue #168). Each row is one unordered pair of standardized ancestry levels plus the rule's severity: columns `ancestry1`, `ancestry2`, and `severity` (`"block"` or `"flag"`), with a header row. Excel (`.xls`/`.xlsx`) and delimited text (`.csv`/`.txt`) files are both accepted, mirroring `readKinshipOverrides`.

Usage

```
readAncestryRules(fileName, sep = ",")
```

Arguments

`fileName`	character vector of length one; path to the rules file (typically the temporary `datapath` from a Shiny file upload).
`sep`	column separator for delimited text files (default `","`).

Details

This reader does not validate structure or domain – that is `checkAncestryRules`'s job. An example rules file expressing the rhesus Indian-origin purity case ships as `example_ancestry_rules.csv` in the package's `extdata/examples` directory.

Value

A data frame of the rows read from `fileName` (typically with columns `ancestry1`, `ancestry2`, and `severity`). Validate it with `checkAncestryRules` before use.

Examples

```
rulesFile <- system.file("extdata", "examples",
  "example_ancestry_rules.csv",
  package = "nprcgenekeepr"
)
rules <- checkAncestryRules(readAncestryRules(rulesFile))
rules
```

---

readKinshipOverrides *Read a kinship overrides table from a file*

Description

Reads an outside-information kinship override table from a user-supplied file into a data frame for `checkKinshipOverrides` and `reportGV`. The expected long form is the output of `kinMatrix2LongForm`: columns `id1`, `id2`, and `kinship`, with a header row, so a user can export the current matrix, edit a few rows, and feed it back. Excel (`.xls`/`.xlsx`) and delimited text (`.csv`/`.txt`) files are both accepted, mirroring `getGenotypes`. Excel files are read with every column as text and then the `kinship` column is converted to numbers, so an Excel file and a delimited text file of the same table give the same result. If any value in that column is not a number, the column is left as text and `checkKinshipOverrides` stops, naming it.

Usage

```
readKinshipOverrides(fileName, sep = ",")
```

Arguments

`fileName`	character vector of length one; path to the override file (typically the temporary `datapath` from a Shiny file upload).
`sep`	column separator for delimited text files (default `","`). Ignored for Excel files.

Details

This reader does not validate structure or domain – that is `checkKinshipOverrides`'s job. `kinship` is the kinship coefficient f, **not** the coefficient of relatedness r ($= 2f$ for non-inbred animals).

Value

A data frame of the rows read from `fileName` (typically with columns `id1`, `id2`, and `kinship`). Validate it with `checkKinshipOverrides` before use.

Examples

```
csvFile <- tempfile(fileext = ".csv")
writeLines(c("id1,id2,kinship", "A1,A2,0.25", "A3,A4,0.125"), csvFile)
overrides <- readKinshipOverrides(fileName = csvFile)
overrides <- checkKinshipOverrides(overrides)
overrides
unlink(csvFile)
```

`readSnapshotHistory` *Read a colony snapshot-history table from a file*

Description

Reads a user-maintained longitudinal colony snapshot history (issue #167) from a file into a data frame for `checkSnapshotHistory`. The history is the single CSV a colony manager keeps between analyses: one row per recorded snapshot in the 27-column version-1 schema (see `checkSnapshotHistory` for the column groups). Excel (`.xls`/`.xlsx`) and delimited text (`.csv`/`.txt`) files are both accepted, mirroring `readKinshipOverrides`. Excel files are read with every column as text and then the numeric columns (e.g. `schemaVersion`) are converted to numbers, so an Excel file and a delimited text file of the same table give the same result. A numeric column that holds a value that is not a number is left as text and `checkSnapshotHistory` stops, naming it.

Usage

```
readSnapshotHistory(fileName, sep = ",")
```

Arguments

`fileName`	character vector of length one; path to the snapshot history file (typically the temporary `datapath` from a Shiny file upload, or a path the user's own scripts maintain).
`sep`	column separator for delimited text files (default `","`). Ignored for Excel files.

Details

This reader does not validate structure or domain — that is `checkSnapshotHistory`'s job, matching the reader/validator sibling-pair convention.

Value

A data frame of the rows read from `fileName`. Validate it with `checkSnapshotHistory` before use.

Examples

```
history <- readSnapshotHistory(system.file("extdata", "examples",
  "example_snapshot_history.csv",
  package = "nprcgenekeepr"
))
history <- checkSnapshotHistory(history)
```

`readTwinRelations` *Read a twin/zygosity relations table from a file*

Description

Reads a user-supplied twin-relations sidecar table into a data frame for `checkTwinRelations`. The expected form is `id1`, `id2`, `code`, with a header row (see `checkTwinRelations` for the accepted `code` values). Excel (`.xls`/`.xlsx`) and delimited text (`.csv`/ `.txt`) files are both accepted, mirroring `readKinshipOverrides` – the closest existing precedent for a validated, pairwise sidecar upload (issue #137 D1/D5, `docs/planning/issue137-twin-zygosity-pedigree-diagram-plan.md`).

Usage

```
readTwinRelations(fileName, sep = ",")
```

Arguments

`fileName`	character vector of length one; path to the twin -relations file (typically the temporary `datapath` from a Shiny file upload).
`sep`	column separator for delimited text files (default `","`). Ignored for Excel files.

Details

This reader does not validate structure or domain – that is `checkTwinRelations`'s job.

Value

A data frame of the rows read from `fileName` (typically with columns `id1`, `id2`, and `code`). Validate it with `checkTwinRelations` before use.

Examples

```
ped <- data.frame(
  id = c("F1", "F2", "S1", "S2"),
  sire = c(NA, NA, "F1", "F1"),
  dam = c(NA, NA, "F2", "F2"),
  sex = c("M", "F", "F", "F"),
  stringsAsFactors = FALSE
)
csvFile <- tempfile(fileext = ".csv")
writeLines(c("id1,id2,code", "S1,S2,MZ twin"), csvFile)
twinRelations <- readTwinRelations(fileName = csvFile)
twinRelations <- checkTwinRelations(ped, twinRelations)
twinRelations
unlink(csvFile)
```

removeAutoGenIds *Remove automatically generated IDs from pedigree*

Description

Identifies automatically generated IDs via `isGeneratedUnknownId()`, the shared detection predicate derived from the configurable auto-ID format (see `getAutoIdFormat`). With the default `"U%04d"`, an automatically generated ID is a capital "U" followed by at least four capital letters or digits (`"U0001"`, or `"U05X3C"` after de-identification). A real animal whose ID merely starts with "U" (`"U1"`, `"U123"`, `"Uma"`) is kept, and so is its place as a sire or dam; a real ID of the full shape (`"U1234"`) is removed unless the pedigree's `placeholder` column (see `qcStudbook`) marks it `FALSE`, and a `TRUE` mark removes a stand-in whatever its ID looks like. The "four or more" match is a prefix, so an ID such as `"U1234abc"` is removed too. Sire and dam entries that are generated IDs are set to `NA`.

Usage

```
removeAutoGenIds(ped)
```

Arguments

`ped`	datatable that is the `Pedigree`. It contains pedigree information. The fields `id`, `sire` and `dam` are required.

Value

A pedigree with automatically generated IDs removed.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
length(examplePedigree$id)
ped <- removeAutoGenIds(examplePedigree)
length(ped$id)
```

removeDuplicates *Remove duplicate records from pedigree*

Description

Part of Pedigree Curation

Usage

```
removeDuplicates(ped, reportErrors = FALSE)
```

Arguments

ped	dataframe that is the Pedigree. It contains pedigree information. The id and recordStatus columns are required.
reportErrors	logical value if TRUE the function returns a character vector of duplicate id values (or NULL when none are found) instead of the de-duplicated pedigree. Only records whose recordStatus is "added" (the records made for parents that have no record of their own) are left out of the search; a record with any other recordStatus, including NA or a blank, is a real animal and is searched.

Details

Returns an updated dataframe with duplicate rows removed.

Returns an error if the table has duplicate IDs with differing data. The error applies only when reportErrors is FALSE. Rows are compared on all columns, including recordStatus.

Value

When reportErrors is FALSE, a dataframe (the Pedigree) with duplicate rows removed; when reportErrors is TRUE, a character vector of duplicate id values, one entry for each extra occurrence of an id (or NULL when none are found).

Examples

```
ped <- nprcgenekeepr::smallPed
newPed <- cbind(ped, recordStatus = rep("original", nrow(ped)))
ped1 <- removeDuplicates(newPed)
nrow(newPed)
nrow(ped1)
pedWithDups <- rbind(newPed, newPed[1:3, ])
ped2 <- removeDuplicates(pedWithDups)
nrow(pedWithDups)
nrow(ped2)
```

---

removeEarlyDates *Remove dates before a specified year*

Description

Dates before a specified year are set to NA. This is often used for dates formed from malformed character representations such as a date in %m-%d-%Y format being read by %Y-%m-%d format

Usage

```
removeEarlyDates(dates, firstYear)
```

Arguments

dates	vector of dates
firstYear	integer value of first (earliest) year in the allowed date range.

Details

NA values are ignored and not changed.

Value

A vector of dates with every date before the year indicated by the numeric value of firstYear set to NA. Dates in firstYear itself are kept.

Examples

```
dates <- structure(c(
  12361, 14400, 15413, NA, 11189, NA, 13224, 10971,
  -432000, 13262
), class = "Date")
cleanedDates <- removeEarlyDates(dates, firstYear = 1000)
dates
cleanedDates
```

---

removePotentialSires *Remove potential sires from a list of IDs*

Description

Remove potential sires from a list of IDs

Usage

```
removePotentialSires(ids, minAge, ped)
```

Arguments

ids	character vector of animal IDs
minAge	integer value giving the inclusive minimum current age (in years) a male must have to be listed as a potential sire. Unlike getPotentialSires, there is no default; it is required.
ped	dataframe that is the Pedigree. It contains pedigree information including the IDs listed in ids.

Value

character vector of Ids with any potential sire Ids removed.

Examples

```
library(nprcgenekeepr)
qcBreeders <- nprcgenekeepr::qcBreeders
pedWithGenotype <- nprcgenekeepr::pedWithGenotype
noSires <- removePotentialSires(
  ids = qcBreeders, minAge = 2,
  ped = pedWithGenotype
)
sires <- getPotentialSires(qcBreeders, ped = pedWithGenotype, minAge = 2)
pedWithGenotype[pedWithGenotype$id %in% noSires, c("sex", "age")]
pedWithGenotype[pedWithGenotype$id %in% sires, c("sex", "age")]
```

---

removeUninformativeFounders

Remove uninformative founders

Description

Founders (having unknown sire and dam) that appear only one time in a pedigree are uninformative and can be removed from a pedigree without loss of information. Removal repeats until no such founder remains, because removing one can leave another founder appearing only once.

Usage

```
removeUninformativeFounders(ped)
```

Arguments

ped	dataframe that is the Pedigree. The id, sire and dam columns are required.

Value

A reduced pedigree.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2, minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
probands <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
```

```
p <- addBackSecondParents(p, ped)
nrow(p)
```

`removeUnknownAnimals` *Remove placeholder animals added for unknown parents*

Description

Remove placeholder animals added for unknown parents

Usage

```
removeUnknownAnimals(ped)
```

Arguments

`ped`	The pedigree information in data.frame format

Value

Pedigree with the animals whose `recordStatus` is `"added"` removed. Every other animal is kept, including one whose `recordStatus` is missing (`NA`) or has any other value. A pedigree without a `recordStatus` column has no animals marked as added, so it is returned unchanged.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
addedPed <- cbind(ped,
  recordStatus = rep("original", nrow(ped)),
  stringsAsFactors = FALSE
)
addedPed[1:3, "recordStatus"] <- "added"
ped2 <- removeUnknownAnimals(addedPed)
nrow(ped)
nrow(ped2)
```

`reportAncestryViolations` *Report ancestry-rule violations within formed groups*

Description

Issue #168 Slice 2: the script-callable flag/inspection half of the ancestry guardrails. Given any list of formed groups (for example the `group` element returned by `groupAddAssign`, excluding its final unused-animals element), a pedigree carrying an `ancestry` column, and a rules table in `checkAncestryRules`'s shape, it reports every within-group pair matching a rule – `block` and `flag` severities alike – plus a rule-coverage summary so permissive silence is visible (a level no rule names participates in no conflict; the summary says how many animals sit at each level and whether any rule covers it).

Usage

```
reportAncestryViolations(groups, ped, rules, overriddenRules = NULL)
```

Arguments

`groups`	list of character vectors of animal IDs, one vector per formed group. `NA` entries (the empty unused-animals marker `groupAddAssign` can produce) are ignored.
`ped`	data frame with at least `id` and `ancestry` columns. Ancestry values are coerced with `toupper(trimws())`, so a post-`qcStudbook` pedigree and a hand-built frame both work.
`rules`	data frame of ancestry compatibility rules; validated here via `checkAncestryRules`.
`overriddenRules`	optional data frame with `ancestry1` and `ancestry2` columns naming the rules overridden for this run (unordered match, case-insensitive). Default `NULL`: no overrides.

Details

A rule listed in `overriddenRules` still reports its violating pairs – with `status "overridden"` rather than `"violation"`, never silently absent. An `overriddenRules` row that matches no rule in `rules` is an error, so a typo cannot silently disable nothing. Any id in `groups` that is absent from `ped$id` stops the function with an error.

Value

A list with two data.frames:

`violations`	One row per violating within-group pair: `group` (integer index into `groups`), `id1`, `id2`, `ancestry1`, `ancestry2` (the two animals' own levels), `rule` (the matched rule's unordered pair as a sorted `"LEVEL-LEVEL"` string), `severity` (`"block"` or `"flag"`), and `status` (`"violation"` or `"overridden"`). Zero rows, same columns, when nothing violates.

coverage One row per standardized ancestry level (CHINESE, INDIAN, HYBRID, JAPANESE, OTHER, UNKNOWN): `ancestry`, `n` (animals in `groups` at that level), and `covered` (`TRUE` when at least one rule names the level).

Examples

```
library(nprcgenekeepr)
ped <- qcStudbook(
  read.csv(
    system.file("extdata", "examples", "example_ancestry_pedigree.csv",
      package = "nprcgenekeepr"
    ),
    stringsAsFactors = FALSE, na.strings = c("", "NA")
  ),
  minSireAge = 2, minDamAge = 2, reportChanges = FALSE, reportErrors = FALSE
)
rules <- checkAncestryRules(readAncestryRules(
  system.file("extdata", "examples", "example_ancestry_rules.csv",
    package = "nprcgenekeepr"
  )
))
reportAncestryViolations(list(c("I1", "C1"), c("I2", "U1")), ped, rules)
```

---

reportGV *Generate a genetic value report for a pedigree*

Description

This is the main function for the Genetic Value Analysis.

Usage

```
reportGV(
  ped,
  guIter = 1000L,
  guThresh = 1L,
  pop = NULL,
  byID = TRUE,
  updateProgress = NULL,
  breedingTable = NULL,
  gestationTable = NULL,
  breedingAgeDefault = NULL,
  gestationDefault = NULL,
  kinshipOverrides = NULL,
  twinRelations = NULL,
  guCutoff = NULL,
  zScoreCutoff = NULL,
  axisPriority = NULL
)
```

Arguments

ped	The pedigree information in data.frame format
guIter	Integer indicating the number of iterations for the gene-drop analysis. Default is 1000 iterations
guThresh	Integer indicating the threshold number of animals for defining a unique allele. Default considers an allele "unique" if it is found in only 1 animal.
pop	Character vector with animal IDs to consider as the population of interest. The default is NULL: an existing `population` column of `ped` is used if there is one, otherwise all animals.
byID	Logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).
updateProgress	Function or NULL. If this function is defined, it will be called once at the start and then once for each animal processed by the gene drop, to update a `shiny::Progress` object.
breedingTable	Optional data.frame overriding the bundled per-species minimum breeding ages used by the unknown-parent mean-kinship correction. `NULL` (the default) uses the bundled `speciesGestation` table.
gestationTable	Optional data.frame overriding the bundled per-species gestation windows used by the correction's conception window. `NULL` uses the bundled table.
breedingAgeDefault	Optional numeric fallback minimum breeding age (years) for species absent from the table. `NULL` uses the built-in 2 years.
gestationDefault	Optional integer fallback gestation window (days) for species absent from the table. `NULL` uses the built-in 210 days.
kinshipOverrides	Optional data.frame of outside-information kinship overrides (`id1`, `id2`, `kinship`; the coefficient *f*, not relatedness *r*) applied to the kinship matrix before mean kinship and the unknown-parent correction. `NULL` (the default) leaves the pedigree-derived matrix unchanged. Ids outside the analysis set are warn-dropped (the run is not aborted). An override REFINES the named kinship cell; it does not suppress the `+ sexMean / 2` unknown-parent correction, which is kept for every animal missing one parent. See `applyKinshipOverrides`.
twinRelations	Optional data.frame of declared twin pairs (`id1`, `id2`, `code`) passed straight through to the internal `kinship` call; only `code == "MZ twin"` rows affect the computed matrix. `NULL` (the default) is a complete no-op, matching today's behavior. See `kinship`'s own `twinRelations` documentation for the full correction mechanism and precondition.
guCutoff	Optional numeric genome-uniqueness threshold passed through to `orderReport` (internal), above which an animal qualifies for the high-uniqueness ranking tier. `NULL` (the default) resolves to `10L`. Not to be confused with `guThresh`, above, which is the unrelated allele-rarity threshold used to *compute* gu itself.

`zScoreCutoff` Optional numeric mean-kinship z-score threshold passed through to `orderReport` (internal), at or below which an animal qualifies for the low-kinship ranking tier. `NULL` (the default) resolves to `0.25`.

`axisPriority` Optional string, one of `"gu"` or `"mk"`, passed through to `orderReport` (internal), naming which axis's cutoff-filter is applied first when ranking. `NULL` (the default) resolves to `"gu"`, today's behavior.

Details

The pedigree `ped` must have the columns `id`, `sire`, `dam`, `gen` (generation number) and `sex`. If any is missing, `reportGV` stops before doing any calculation, with a message naming every missing column.

The population of interest (see `pop`) must hold at least 2 animals, because the report ranks animals against one another. With fewer, `reportGV` stops before doing any calculation, with a message giving the number of animals.

Reported genome uniqueness (`gu`) is set to 0 for "Undetermined" animals – those with both parents unknown (U-id aware) and no recorded origin – because their apparent uniqueness is an artifact of unknown parentage (decline-to-credit policy). Imports (both parents unknown but with a recorded origin) and all other animals are unaffected, and `calcGU` itself is unchanged.

Value

An object of class `nprcgenekeeprGV`: a list with elements `report` (a dataframe with the genetic value report, with animals ranked in order of descending value; it carries both a `gu` column and a `guSE` column – the Monte Carlo sampling standard error of each genome uniqueness estimate, see `calcGUSE`), `kinship` (the kinship matrix), `gu` (a dataframe with a `gu` column of genome uniqueness values and a `guSE` column of their standard errors; `gu` and `guSE` are reported as 0 for unknown-origin both-unknown "Undetermined" animals, whose apparent uniqueness is an artifact of unknown parentage), `fe` (founder equivalents), `fg` (founder genome equivalents), `fgSE` (the Monte Carlo sampling standard error of `fg`, computed from the same gene drop as `fg`; a single colony-level number, `NA` when a contributing founder is retained in zero gene-drop iterations – see `calcFGSE`), `neGD` (gene diversity retained in the founding gene pool, `1 - 1 / (2 * fg)`; a colony-level scalar beside `fg`, `NA` when `fg` is `NA`; see `calcGeneDiversity`), `neSexRatio` (the demographic sex-ratio effective size, `4 * nMale * nFemale / (nMale + nFemale)`, over the current living breeders – a different population than `fg` and `neGD`; `0` when a breeding sex is absent; see `calcNeSexRatio`), `neVariance` (the variance effective size, the mean-adjusted Crow-Kimura form `(N * kbar - 1) / (kbar - 1 + Vk / kbar)` over the same current living breeders; `NA` when fewer than two living breeders; see `calcNeVariance`), `maleFounders` and `femaleFounders` (dataframes of the known male and female founder records), `nMaleFounders` and `nFemaleFounders` (the counts of those founders), and `total` (the total number of known founders). `fg`, `fgSE` and `neGD` come from a gene drop that ignores genotypes; it is the same gene drop that gives `gu` when the pedigree has none. The founder lists hold only *known* founders: an animal with both parents unknown whose id was generated to stand in for an unknown parent is left out, because that placeholder may be a known founder, another animal in the population, or an animal outside it. `fe` and `fg`, by contrast, count every animal with both parents unknown as a separate, unrelated founder (see `calcFEFG`), so they can differ from `total`. `report` also carries a `flagged` column (issue #127): `TRUE` for a one-unknown-parent animal left uncorrected by `correctUnknownParentMeanKinship()` for lack of an eligible breeding-age peer

cohort (or a missing birth date); FALSE for every other animal, including one-unknown-parent animals that were successfully corrected and all fully-known/both-unknown animals (which are never candidates for this correction).

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2, minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped,
  removeUninformative = FALSE,
  addBackParents = FALSE
)
## guIter = 50 only keeps the example fast. With so few iterations `fg` and
## `fgSE` can be degenerate, and a warning about that may be printed.
geneticValue <- reportGV(ped,
  guIter = 50, # should be >= 1000
  guThresh = 3,
  byID = TRUE,
  updateProgress = NULL
)
trimmedGeneticValue <- reportGV(trimmedPed,
  guIter = 50, # should be >= 1000
  guThresh = 3,
  byID = TRUE,
  updateProgress = NULL
)
rpt <- trimmedGeneticValue[["report"]]
kmat <- trimmedGeneticValue[["kinship"]]
f <- trimmedGeneticValue[["total"]]
mf <- trimmedGeneticValue[["maleFounders"]]
ff <- trimmedGeneticValue[["femaleFounders"]]
nmf <- trimmedGeneticValue[["nMaleFounders"]]
nff <- trimmedGeneticValue[["nFemaleFounders"]]
fe <- trimmedGeneticValue[["fe"]]
fg <- trimmedGeneticValue[["fg"]]
```

`reportMatePairs` *Report eligible individual mate pairs with kinship and genetic-value context*

Description

Issue #151 Slice 1: a report of opposite-sex, minimum-age-eligible mate-pair candidates. Its pairs are those kinMatrix2LongForm followed by filterPairs give, in the same order, built directly, so the table of every pair (same-sex pairs included) is never made; filterKinMatrix scopes the population. Each row is optionally enriched with marker-based kinship (markerKinship) and per-parent genetic-value context (reportGV's indivMeanKin/gu) when available. No composite/blended ranking score is computed – callers sort/filter the raw columns themselves (see docs/planning/issue151-individual-mate-pair-analysis-plan.md Section 3, decision D3).

Usage

```
reportMatePairs(
  ped,
  kmat,
  markerKmat = NULL,
  geneticValues = NULL,
  minAge = 1L,
  populationIds = NULL,
  exclude = character(0L),
  ancestryRules = NULL,
  overriddenRules = NULL
)
```

Arguments

ped	Pedigree data.frame with at least id, sire, dam, sex, and age columns.
kmat	Pedigree kinship matrix (e.g. as returned by kinship), id x id, covering every id considered.
markerKmat	Optional genotype-only KING-robust kinship matrix (e.g. as returned by markerKinship), id x id, which may cover only a subset of ped$id (not every animal is genotyped). NULL (the default) leaves markerKinship as NA for every pair.
geneticValues	Optional reportGV-shaped list (only $report, with id, indivMeanKin, and gu columns, is used). NULL (the default) leaves the four per-parent genetic-value columns NA for every pair.
minAge	Numeric scalar, the minimum age (in the same units as ped$age) for an individual to be pair-eligible. Default 1, mirroring Breeding Groups' own scalar convention. A missing (NA) age passes this screen (see Details) – use populationIds to actually bound the candidate population.
populationIds	Optional character vector of ids to which the analysis is scoped, applied before the pair-reshape (D4, see Details). NULL (the default) performs no scoping; character(0) scopes to nobody and returns a zero-row result.
exclude	Character vector of ids to drop entirely, regardless of any other eligibility screen. Default character(0) (no caller-supplied exclusions).
ancestryRules	Optional data frame of ancestry compatibility rules in checkAncestryRules's shape (ancestry1, ancestry2, severity); validated here. ped must then carry an ancestry column, or the call stops. NULL (the default) applies no ancestry

screen. A valid zero-rule table is accepted: the result then has the ancestry columns, with nothing moved.

`overriddenRules`
: Optional data frame with `ancestry1` and `ancestry2` columns naming `block` rules to override for this call (unordered, case-insensitive match, as in `reportAncestryViolations`); a `reason` column is accepted and ignored. An override naming no rule in `ancestryRules`, a `flag` rule, or the same rule twice is an error, as is giving overrides without `ancestryRules`. `NULL` or zero rows: none.

Details

Population scoping (D4). `minAge` alone does not bound the candidate-pair table on real, imperfectly-curated data: a missing age *passes* `filterAge()`'s screen rather than being excluded by it (see that function's own semantics), so long-retired or never-dated individuals can still appear. `populationIds`, applied via `filterKinMatrix` *before* the pair-reshape, is the mechanism that actually bounds both correctness and table size; the default `NULL` performs no scoping (every individual in `kmat` is a candidate). An explicit `character(0)` is a valid, distinct input – "scope to nobody" – and returns a zero-row `pairs` frame with the full column shape, not an error.

Sex. Only animals recorded as male (`"M"`) or female (`"F"`) are paired. An animal with any other sex (unknown, hermaphrodite or missing) is left out before pairs are formed and appears in neither `pairs` nor `excluded`; the Mate Pair Analysis tab says how many animals were left out.

Exclusion transparency (D5). Two independent screens produce `excluded` rows: the age screen (`"under minimum age"`) and the caller-supplied `exclude` id list (`"user-excluded"`). A pair failing the age screen is reported with that reason and never reaches the user-exclude screen, so each dropped pair carries exactly one reason from this closed, enumerable vocabulary.

Graceful degradation. `markerKmat` and `geneticValues` are both `NULL`-safe: when absent (or when a specific pair is not covered by them), the corresponding columns are `NA` for that row – the pair itself is never dropped or the call errored, mirroring `modMarkerGeneticsServer`'s own "not yet uploaded" contract.

Ancestry rules (issue #169). `ancestryRules` applies the same center-configurable rules table as the Breeding Groups ancestry guardrails (`checkAncestryRules`) to individual pairs; a rule matches a pair whichever animal is the sire and whichever the dam. A `block` rule moves the matching pair out of `pairs` and into `excluded` with the reason `"ancestry rule"` (here "block" means "not listed as eligible", not "never placed together"); a `flag` rule keeps the pair in `pairs` and annotates it. Naming a block rule in `overriddenRules` keeps its pairs in `pairs`, marked `"overridden"` – never silently absent. The ancestry screen runs last: a pair that already failed the age or user-exclude screen keeps that reason. Rules only move and label pairs; every pair is either in `pairs` or in `excluded`, with or without rules. An animal whose ancestry no rule names, or is missing, matches nothing. With `ancestryRules = NULL` (the default) the result is unchanged, and `ped` needs no `ancestry` column.

Value

A list with two data.frames (three when `ancestryRules` is given):

`pairs`
: One row per eligible opposite-sex pair surviving every screen: `sireId`, `damId`, `kinship`, `markerKinship` (`NA` if unavailable), `sireIndivMeanKin`, `sireGu`, `damIndivMeanKin`, `damGu` (the latter four `NA` if `geneticValues` is not supplied or the parent is absent

from its report). With ancestryRules, three columns follow: ancestryRule (the matched rule as a sorted "LEVEL-LEVEL" string), ancestrySeverity ("flag" or "block") and ancestryStatus ("violation", or "overridden" for an overridden block rule), all NA for a pair no rule matches.

excluded — One row per pair dropped by the age, user-exclude or ancestry screen: sireId, damId, reason ("under minimum age", "user-excluded" or, with ancestryRules, "ancestry rule"). With ancestryRules an ancestryRule column follows: the matched rule for an ancestry exclusion, NA otherwise.

ancestryCoverage — Only with ancestryRules: one row per standardized ancestry level (CHINESE, INDIAN, HYBRID, JAPANESE, OTHER, UNKNOWN) with ancestry, n (distinct animals at that level in pairs and excluded) and covered (TRUE when a rule names the level), so a level no rule covers is visible.

Examples

```
library(nprcgenekeepr)
ped <- data.frame(
  id = c("S1", "D1", "A", "B"),
  sire = c(NA, NA, "S1", NA),
  dam = c(NA, NA, "D1", NA),
  sex = c("M", "F", "M", "F"),
  age = c(10, 10, 5, 5),
  stringsAsFactors = FALSE
)
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
result <- reportMatePairs(ped, kmat, minAge = 1)
result$pairs

# Ancestry rules (issue #169): the example rules block INDIAN x CHINESE
# and INDIAN x HYBRID pairs and flag INDIAN x UNKNOWN / INDIAN x OTHER
ancPed <- qcStudbook(
  read.csv(
    system.file("extdata", "examples", "example_ancestry_pedigree.csv",
      package = "nprcgenekeepr"
    ),
    stringsAsFactors = FALSE, na.strings = c("", "NA")
  ),
  minSireAge = 2, minDamAge = 2, reportChanges = FALSE,
  reportErrors = FALSE
)
ancPed$gen <- findGeneration(ancPed$id, ancPed$sire, ancPed$dam)
ancKmat <- kinship(ancPed$id, ancPed$sire, ancPed$dam, ancPed$gen)
rules <- checkAncestryRules(readAncestryRules(
  system.file("extdata", "examples", "example_ancestry_rules.csv",
    package = "nprcgenekeepr"
  )
))
ancResult <- reportMatePairs(ancPed, ancKmat, ancestryRules = rules)
ancResult$excluded
```

```
ancResult$ancestryCoverage
# overriding one block rule keeps its pairs listed, marked "overridden"
overridden <- reportMatePairs(ancPed, ancKmat,
  ancestryRules = rules,
  overriddenRules = data.frame(ancestry1 = "CHINESE", ancestry2 = "INDIAN")
)
overridden$pairs[
  which(overridden$pairs$ancestryStatus == "overridden"),
  c("sireId", "damId", "ancestryRule", "ancestryStatus")
]
```

`resolveCrossCenterIds` *Merge two centers' pedigrees via a curator-confirmed identity link*

Description

Collapses a transferred animal's two center-specific records into ONE node with its real parents intact, instead of leaving it as an artificial founder at the receiving center – the failure mode issue #130 names directly (a transferred animal loses its recorded lineage because the two centers use independent id namespaces). Follows the `getPedigreeSource()` design style (D5): a deterministic, curator-supplied cross-reference table, never coincidental same-string ids, and fail-loud validation on any ambiguity.

Usage

```
resolveCrossCenterIds(pedA, pedB, mapping)
```

Arguments

`pedA`	a pedigree data.frame for the first center, with (at least) columns `id`, `sire`, and `dam`.
`pedB`	a pedigree data.frame for the second center, with (at least) columns `id`, `sire`, and `dam`.
`mapping`	a data.frame with columns `idA` and `idB`: one row per curator-confirmed cross-center identity link, naming the same physical animal's id in `pedA` and in `pedB`. Each id may appear at most once in `idA` and at most once in `idB`.

Details

For each `mapping` row, the two records collapse into one, keyed by the `idA` value (the canonical id): every reference to the mapped `idB` value anywhere in `pedB` – as that animal's own `id` or as a `sire`/`dam` pointer on any other animal – is rewritten to `idA`. The merged individual's `sire`/`dam` prefer whichever side has a non-`NA` value (this is what fixes the artificial-founder problem: a center that never knew the animal's real parents recorded `NA`, and the origin center's real parents win). A mapped pair whose two sides both record a non-`NA`, *different* `sire` or `dam` is a real data inconsistency, not something to silently pick a side on, so it errors instead. Animals not named in `mapping` pass through unchanged; an id string present in both `pedA` and `pedB` that is *not* declared in `mapping` is also

an error – per D5, identity is established only by the explicit mapping table, never assumed from a coincidentally matching id string across the two centers' independent namespaces. So is an animal of pedB with no mapping row whose id is one mapping gives a pedA animal: the merged animal is keyed by that idA value, so the merge would hold two rows with the same id. (The mirror case, a pedA animal with no mapping row whose id is a mapped idB value, is not an error: the mapped pedB animal is renamed to its idA value and every id stays unique.) As of issue #149 Slice 1 (D10), a merged pair's other columns (beyond id/ sire/dam) follow the identical non-NA-preferred/ error-on-conflict rule, and a column only one center recorded is kept as that center recorded it – previously these were silently dropped. The one exception is the placeholder mark (see qcStudbook): a stand-in for an unknown parent that a curator links to a known animal is that animal, so the merged animal is real (FALSE) when either center recorded it as real, a stand-in (TRUE) when both did, and a mark on only one side is kept; a disagreement is not an error.

Value

A single merged pedigree data.frame over the union of pedA's and pedB's columns, with one row per distinct animal (mapped pairs collapsed to their canonical idA id). Rows are ordered as the pedA rows not in the mapping, then the merged mapped pairs, then the pedB rows not in the mapping.

See Also

getFileDirectRelatives, checkCrossCenterMapping

Examples

```
library(nprcgenekeepr)
pedA <- data.frame(
  id = c("P1", "P2", "T1"), sire = c(NA, NA, "P1"), dam = c(NA, NA, "P2"),
  stringsAsFactors = FALSE
)
## X9 is the SAME physical animal as T1, but Center B recorded it as an
## artificial founder because it never knew the real parents.
pedB <- data.frame(
  id = c("X9", "O1"), sire = c(NA, "X9"), dam = c(NA, NA),
  stringsAsFactors = FALSE
)
mapping <- data.frame(idA = "T1", idB = "X9", stringsAsFactors = FALSE)
resolveCrossCenterIds(pedA, pedB, mapping)
```

---

rhesusGenotypes *Rhesus genotypes (two haplotypes per animal)*

Description

A dataframe with two haplotypes per animal.

Usage

```
data(rhesusGenotypes)
```

Format

An object of class `data.frame` with 31 rows and 3 columns.

Details

There are 31 rows and 3 columns.

Represents 31 animals that are also in the obfuscated `rhesusPedigree` pedigree from *obfuscated_rhesus_mhc_breeder_geno*

id – character column of animal IDs

first_name – a generic name for the first haplotype

second_name – a generic name for the second haplotype

Examples

```
library(nprcgenekeepr)
data("rhesusGenotypes")
```

`rhesusPedigree` *Obfuscated rhesus pedigree object*

Description

A pedigree object. Represents an obfuscated pedigree from *obfuscated_rhesus_mhc_ped.csv* where the IDs and dates have been modified to de-identify the data.

id – character column of animal IDs

sire – the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam – the female parent of the animal indicated by the `id` column. Unknown dams are indicated with `NA`

sex – factor with levels: "F", "M". Sex specifier for an individual.

gen – generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth – `Date` vector of birth dates

exit – `Date` vector, all `NA` (no exit dates are recorded in this obfuscated pedigree)

age – numerical vector of age in years

Usage

```
data(rhesusPedigree)
```

Format

An object of class `data.frame` with 375 rows and 8 columns.

Examples

```
library(nprcgenekeepr)
data("rhesusPedigree")
```

`runGeneKeepR` *Run the GeneKeepR Shiny Application*

Description

Launches the GeneKeepR Shiny application. It uses a module-based architecture with a Home tab and improved UI components. The call blocks the R session until the application is stopped.

Usage

```
runGeneKeepR(port = 6013L, launch.browser = TRUE)
```

Arguments

`port`	Integer port number for the Shiny server (default 6013)
`launch.browser`	Logical; whether to launch browser (default TRUE)

Details

The application has 16 top-level tabs (15 when the ORIP Reporting tab is hidden) and a "More" menu (Settings, About, Help):

- Home tab with navigation buttons
- Input tab with enhanced QC display, plus dynamic error and changed columns tabs
- Pedigree Browser with focal animal support and a Diagram tab
- Age-Sex Pyramid with enhanced controls
- Genetic Value Analysis with visualizations
- Summary Statistics with popovers
- ORIP Reporting (shown only for the ONPRC site configuration)
- Breeding Groups with group panels
- Mate Pair Analysis
- Genetic Diversity
- Marker Genetics
- Cross-Center Identity
- De-Identified Export
- Potential Parents
- Genetic Value Analysis and Breeding Group Description
- Genetic-Health Trends

`runModularApp` is a soft-deprecated alias for this function.

Value

Called for its side effect; blocks until the app is stopped. Returns, invisibly, the value passed to `shiny::stopApp()` (normally `NULL`).

See Also

`runModularApp`, a soft-deprecated alias for this function.

Examples

```
## Not run:
library(nprcgenekeepr)
runGeneKeepR()

## End(Not run)
```

---

runModularApp *Run the Modular Version of GeneKeepR (Deprecated)*

Description

`runModularApp()` has been renamed to `runGeneKeepR`, a name that says what the function does. `runModularApp()` is now a soft-deprecated alias that launches the application via `runGeneKeepR`, passing all arguments through. Existing callers continue to work. Called directly, it emits a lifecycle message ("was deprecated in nprcgenekeepr 2.0.0"); soft deprecation is silent for calls made from inside another package.

Usage

```
runModularApp(port = 6013L, launch.browser = TRUE)
```

Arguments

`port`	Integer port number for the Shiny server (default 6013)
`launch.browser`	Logical; whether to launch browser (default TRUE)

Value

Called for its side effect; blocks until the app is stopped. Returns, invisibly, the value passed to `shiny::stopApp()` (normally `NULL`), as `runGeneKeepR` does.

See Also

`runGeneKeepR`, the function this now launches.

Examples

```
## Not run:
library(nprcgenekeepr)
runModularApp()

## End(Not run)
```

`runQcStudbook` *Run Quality Control on Studbook with UI-Friendly Results*

Description

Wrapper function that runs `qcStudbook` and processes results into a format suitable for Shiny UI display. This function performs two passes: first to check for errors, then to get the cleaned data if no errors exist.

Usage

```
runQcStudbook(
  ped,
  minSireAge = NULL,
  minDamAge = NULL,
  minParentAge = lifecycle::deprecated(),
  reportChanges = FALSE
)
```

Arguments

`ped`	data.frame containing pedigree data with columns including id, sire, dam, sex, and birth (all required), and optionally death, departure, etc.
`minSireAge`	numeric minimum age in years for a male to have sired an offspring. `NULL` (default) looks up each sire's species floor via `getSpeciesMinBreedingAge` (2 years when species is unknown); a supplied value overrides that floor.
`minDamAge`	numeric minimum age in years for a female to have borne an offspring. `NULL` (default) looks up each dam's species floor via `getSpeciesMinBreedingAge` (2 years when species is unknown); a supplied value overrides that floor.
`minParentAge`	**[Deprecated]** Deprecated scalar minimum parent age. Supplying it sets both `minSireAge` and `minDamAge`; use those sex-specific parameters instead.
`reportChanges`	logical whether to put notices in the result (default FALSE). When TRUE, `changedCols` and `hasChangedCols` in the qcResult report all renamed columns, while `warnings` lists the case and space changes and, when animals in the cleaned pedigree have no birth date, one warning, "Animals with no birth date", that says how many. When animals have an exit date earlier than their birth date, one more warning, "Animals with a death or departure before birth", says how many and names the first five. These warnings are notices only: they never make `hasErrors` TRUE, and they are left out when errors were found, because no cleaned pedigree exists then. When FALSE, `warnings` is empty.

Value

A list with the following components:

- `cleaned` - The cleaned pedigree data.frame with standardized column names, added generation numbers, etc. NULL if errors were found.
- `qcResult` - Result from `processQcStudbookResult` containing errors, warnings, changedCols, hasErrors, and hasChangedCols.
- `errorLst` - The raw `nprcgenekeeprErr` list from `qcStudbook`'s first pass (the same object `qcResult` was derived from), exposed so callers that need the raw fields (e.g. `femaleSires`, `failedDatabaseConnection`) do not have to call `qcStudbook` a second time themselves.

See Also

`qcStudbook` for the underlying QC function

`processQcStudbookResult` for result processing

`modInputServer` for Shiny module integration

Examples

```
data("pedGood", package = "nprcgenekeepr")
result <- runQcStudbook(pedGood, minSireAge = 2.0, minDamAge = 2.0)
if (!result$qcResult$hasErrors) {
  cleanedPed <- result$cleaned
}
```

---

safeExecute *Execute an expression with error handling*

Description

Executes an expression with comprehensive error handling. On error, logs the error and returns a default value instead of stopping execution. This is particularly useful in Shiny reactive contexts where errors should be handled gracefully.

Usage

```
safeExecute(
  expr,
  module = "unknown",
  default = NULL,
  silent = FALSE,
  notify = FALSE
)
```

Arguments

expr	An expression to evaluate.
module	character. Name of the calling module for logging purposes.
default	The value to return if an error occurs. Defaults to NULL.
silent	logical. If TRUE, suppresses the error and warning logging. Defaults to FALSE.
notify	logical. If TRUE and in a Shiny context, shows a notification to the user. Defaults to FALSE.

Details

A warning is also caught. It is logged (unless `silent` is TRUE) and `expr` is then evaluated a second time with warnings suppressed, so `expr` runs twice when it signals a warning. The value from the second evaluation is returned.

Value

The result of evaluating `expr`, or `default` if an error occurs.

See Also

`logModuleEvent` for logging

Examples

```
# Returns 4
safeExecute({ 2 + 2 }, module = "test")

# Returns NULL and logs error
safeExecute({ stop("Error!") }, module = "test")

# Returns custom default on error
safeExecute({ stop("Error!") }, module = "test", default = data.frame())
```

---

saveDataframesAsFiles *Write copy of dataframes to either CSV, TXT, or Excel file*

Description

Takes a list of dataframes and creates a file based on the list name of the dataframe and the extension for the file type. The extension is the value of `fileType` itself, so Excel files get the extension `.excel`, not `.xlsx`. The list must be named; an unnamed list gives files without a base name.

Usage

```
saveDataframesAsFiles(dfList, baseDir, fileType = "csv")
```

Arguments

dfList	named list of dataframes to be stored as files. The names of the list supply the file names.
baseDir	character vector of length one with the directory path. The directory must already exist.
fileType	character vector of length one with possible values of "txt", "csv", or "excel". Default value is "csv".

Value

A character vector of the path names of the files saved, built as baseDir, a slash, the list name, a period and fileType. baseDir is used exactly as supplied, so a relative path stays relative.

Examples

```
library(nprcgenekeepr)
dfList <- list(
  lacy1989Ped = nprcgenekeepr::lacy1989Ped,
  pedGood = nprcgenekeepr::pedGood
)
## Write each data frame to a CSV file under a temporary directory.
files <- saveDataframesAsFiles(dfList,
  baseDir = tempdir(), fileType = "csv"
)
basename(files)
```

---

savePlotToFile *Save Plot to File*

Description

Helper function to save ggplot2 plots to files with consistent settings and error handling. Supports PNG, PDF, and SVG formats with configurable dimensions and resolution.

Usage

```
savePlotToFile(
  plot,
  file,
  format = NULL,
  width = 8L,
  height = 6L,
  dpi = 150L,
  units = "in",
  bg = "white"
)
```

Arguments

plot	A ggplot2 plot object to save. If NULL or not a ggplot2 object, returns FALSE.
file	character. The file path to save the plot to.
format	character. Output format, such as "png", "pdf", "svg", "jpg", "jpeg", or "tiff". Defaults to NULL, in which case the format is inferred from the file extension, falling back to "png" when the extension is not recognized.
width	numeric. Plot width in inches. Defaults to 8.
height	numeric. Plot height in inches. Defaults to 6.
dpi	numeric. Resolution in dots per inch for raster formats. Defaults to 150 for good quality web/screen use. Use 300 for print.
units	character. Units for width and height. Defaults to "in" (inches).
bg	character. Background color. Defaults to "white".

Value

Logical. TRUE if the file was saved successfully, FALSE otherwise.

See Also

ggsave for the underlying save function

Examples

```
library(ggplot2)
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
pngFile <- tempfile(fileext = ".png")
pdfFile <- tempfile(fileext = ".pdf")
hiResFile <- tempfile(fileext = ".png")
savePlotToFile(p, pngFile)
savePlotToFile(p, pdfFile, format = "pdf")
savePlotToFile(p, hiResFile, dpi = 300)
unlink(c(pngFile, pdfFile, hiResFile))
```

---

setAutoIdFormat *Set the auto-generated unknown-ID format*

Description

Sets the sprintf template used to mint and detect placeholder IDs for unknown parents (see addUIds). The format must have a non-empty literal prefix before its first "%" (used for detection) and must consume a single integer (used for generation), e.g. "U%04d" or "AUTO%05d". The number must print as capital letters or digits, so that the IDs the format makes are recognized as placeholders (see removeAutoGenIds); a format that prints lowercase letters ("U%04x") or pads with spaces ("U%4d") is refused. The setting is stored in options(nprcgenekeepr.autoIdFormat=) and read by getAutoIdFormat.

Usage

```
setAutoIdFormat(format)
```

Arguments

format	A single character string: the auto-ID sprintf format.

Value

The previous format, returned invisibly.

See Also

getAutoIdFormat, addUIds

Examples

```
old <- setAutoIdFormat("AUTO%05d")
getAutoIdFormat()
setAutoIdFormat(old) # restore
```

---

setExit *Set the exit date when no exit column exists*

Description

Part of Pedigree Curation

Usage

```
setExit(ped, timeOrigin = as.Date("1970-01-01"))
```

Arguments

ped	dataframe of pedigree and demographic information potentially containing columns indicating the birth and death dates of an individual. The table may also contain dates of sale (departure). Optional columns are birth, death, and departure.
timeOrigin	date object used by as.Date to set origin.

Details

An exit column is added only when the pedigree has a birth column and no exit column. When both death and departure are present the exit is the earlier of the two; it is NA if neither column exists. The column names are matched without regard to case (Death is read as death), and the new column is named exit. A pedigree without a birth column, or with no rows, is returned unchanged.

Value

A dataframe with an updated pedigree with exit dates specified based on date information that was available.

Examples

```
library(lubridate)
library(nprcgenekeepr)
death <- mdy(paste0(
  sample(1:12, 10, replace = TRUE), "-",
  sample(1:28, 10, replace = TRUE), "-",
  sample(seq(0, 15, by = 3), 10, replace = TRUE) + 2000
))
departure <- as.Date(rep(NA, 10), origin = as.Date("1970-01-01"))
departure[c(1, 3, 6)] <- as.Date(death[c(1, 3, 6)],
  origin = as.Date("1970-01-01")
)
death[c(1, 3, 5)] <- NA
death[6] <- death[6] + days(1)
ped <- data.frame(
  id = paste0(100 + 1:10),
  birth = mdy(paste0(
    sample(1:12, 10, replace = TRUE), "-",
    sample(1:28, 10, replace = TRUE), "-",
    sample(seq(0, 20, by = 3), 10, replace = TRUE) + 1980
  )),
  death = death,
  departure = departure,
  stringsAsFactors = FALSE
)
pedWithExit <- setExit(ped)
```

---

setLabKeyDefaults *Configure Rlabkey authentication for the current session*

Description

Sets up the credentials that `getDemographics` (and any other `Rlabkey` call) uses to authenticate against the LabKey EHR server. An API key, when available, is preferred; otherwise the function falls back to a `.netrc`/`_netrc` file; when neither is present it stops with an actionable error rather than letting `Rlabkey` fail later with an opaque message.

Usage

```
setLabKeyDefaults(siteInfo = getSiteInfo())
```

Arguments

`siteInfo`	list of site information as returned by `getSiteInfo`. The elements used are `baseUrl`, `configFile`, `homeDir`, and `sysname`.

Details

The API key is sourced, in order of precedence, from

1. the environment variable `NPRCGENEKEEPR_LABKEY_APIKEY`, then
2. an `apiKey` entry in the nprcgenekeepr configuration file.

When an API key is found, `labkey.setDefaults` is called with that key and `siteInfo$baseUrl`. When no API key is found, the function checks for a netrc file (the `NETRC` environment variable, then the home-directory `.netrc` on non-Windows or `_netrc` on Windows) and, if present, leaves `Rlabkey` to use it. The API key is never read from or written to the package sources; keep it in the environment, the configuration file, or the netrc file only.

Value

Invisibly, a list with elements `method` (one of `"apiKey"` or `"netrc"`) and `baseUrl`. Stops with an error when no credential can be found.

Examples

```
## Not run:
## Requires an apiKey (env var or config) or a .netrc file to succeed.
library(nprcgenekeepr)
result <- tryCatch(
  setLabKeyDefaults(getSiteInfo(expectConfigFile = FALSE)),
  error = function(e) conditionMessage(e)
)

## End(Not run)
```

---

setPopulation *Flag animals as the population of interest*

Description

Part of the pedigree filtering toolset.

Usage

```
setPopulation(ped, ids)
```

Arguments

`ped`	datatable that is the `Pedigree`. It contains pedigree information. The `id` column is required.
`ids`	character vector of IDs to be flagged as part of the population under consideration.

Value

An updated pedigree with the `population` column added or updated by being set to `TRUE` for the animal IDs in `ped$id` and `FALSE` otherwise. If `ids` is empty, all animals are flagged `TRUE`.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2, minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
breederPed <- setPopulation(ped = breederPed, ids = focalAnimals)
nrow(breederPed[breederPed$population, ])
```

---

set_seed *Set a reproducible RNG seed across R versions*

Description

R 3.6 changed how `set.seed` and `sample` work. For R 3.6 and later this function calls `set.seed(seed, sample.kind = "Rounding")`, which restores the earlier `sample()` behavior, so results of `sample()` differ from those after a plain `set.seed(seed)`. For earlier versions of R it calls `set.seed(seed)`. This lets unit tests give the same results on multiple versions of R in a CICD test build. Messages and warnings from `set.seed` are suppressed.

Usage

```
set_seed(seed = 1L)
```

Arguments

`seed`	argument to `set.seed`

Value

NULL, invisibly.

Examples

```
set_seed(1)
rnorm(5)
```

shouldShowChangedColsTab *Determine if Changed Columns tab should be displayed*

Description

Checks the changedCols list to determine if the Changed Columns tab should be inserted into the application navigation. The tab is shown when column names were modified during QC processing.

Usage

```
shouldShowChangedColsTab(changedCols)
```

Arguments

changedCols	list containing information about changed column names. Expected fields include: caseChange, spaceRemoved, periodRemoved, underScoreRemoved, egoToId, egoidToId, sireIdToSire, damIdToDam, birthdateToBirth, deathdateToDeath, recordstatusToRecordStatus, fromcenterToFromCenter, geographicoriginToGeographicOrigin

Value

Logical. TRUE if columns were changed and tab should be shown, FALSE otherwise.

See Also

checkChangedColsLst for the original implementation

Examples

```
library(nprcgenekeepr)
## No column changes recorded: the tab stays hidden.
shouldShowChangedColsTab(list())
## A recorded case change: the tab should be shown.
shouldShowChangedColsTab(list(caseChange = c(Id = "id")))
```

shouldShowOripTab *Determine if the ORIP Reporting tab should be displayed*

Description

Determines whether the Oregon (ONPRC)-specific ORIP Reporting tab should be shown in the application navigation. ORIP (Office of Research Infrastructure Programs) grant reporting is specific to ONPRC, so the tab is shown only when an actual site configuration file is present *and* it identifies the colony as ONPRC. The getSiteInfo default fallback (center = "ONPRC" when no configuration file exists) does NOT show the tab.

Usage

```
shouldShowOripTab(center, hasConfigFile)
```

Arguments

center	Character scalar naming the colony center, as returned by `getSiteInfo()$center` (e.g. "ONPRC" or "SNPRC"). `NULL`, missing values, or any value other than "ONPRC" yield FALSE.
hasConfigFile	Logical scalar indicating whether an actual site configuration file is present (e.g. `file.exists(getSiteInfo()$configFile)`). When FALSE the colony center is the default fallback and the tab is not shown.

Value

Logical. TRUE if a real ONPRC configuration is active and the tab should be shown, FALSE otherwise.

See Also

`getSiteInfo` for the site configuration source and `shouldShowChangedColsTab` for the sibling tab-visibility predicate.

Examples

```
library(nprcgenekeepr)
shouldShowOripTab("ONPRC", TRUE)  # TRUE
shouldShowOripTab("SNPRC", TRUE)  # FALSE
shouldShowOripTab("ONPRC", FALSE) # FALSE (default fallback, no config file)
```

---

shrinkPedigree *Shrink a pedigree to fit within a bit-size budget*

Description

Trims a pedigree down to the individuals needed to keep it genetically informative within a genotyping-cost budget (`maxBits`), given which individuals are genotyped (`genotyped`) and, optionally, which are affected by a trait of interest (`affected`). It is equivalent to `kinship2::pedigree.shrink()`, working on this package's `id`/`sire`/`dam` data-frame pedigree. Three tiers are applied in order:

1. **Unavailable trim.** Iteratively removes terminal (leaf) individuals who are not genotyped, then removes any founder couple with exactly one child together and no other mate, when both parents are themselves founders and neither is genotyped (the couple's shared child is promoted to founder status rather than removed), then removes any remaining childless founder ("stray marry-in") *regardless of genotyped status*.

2. **Non-informative trim.** Removes a genotyped, non-parent individual whose own `sire` and `dam` are both known and both genotyped, when the individual is not `affected` (an `NA` `affected` status counts as unaffected here) – they add no genotype information beyond what their parents already supply. An individual with only one known parent (one of `sire`/`dam` known, the other `NA`) is never trimmed by this tier. Such partial parentage is ordinary data in this package (see `getIdsWithOneParent`), whereas kinship2 does not allow it.
3. **Affected-priority trim.** While the pedigree's bit size (`2 * nNonFounder - nFounder`) still exceeds `maxBits`, removes one genotyped, non-parent individual at a time – trying `NA`-affected candidates first, then unaffected, then affected – choosing whichever single candidate's removal (including any cascade through tiers 1-2 above) minimizes the resulting bit size. Ties are broken deterministically by lowest `id`, compared as a string, so the same input always gives the same answer (kinship2 breaks ties at random). `idTrimmed` and `idList$affected` record every id actually removed each round, including those removed by a cascade, so `pedSizeOriginal - pedSizeFinal` always equals `length(idTrimmed)`.

Usage

```
shrinkPedigree(ped, genotyped, affected = NULL, maxBits = 16L)
```

Arguments

`ped`	a pedigree `data.frame`. The fields `id`, `sire` and `dam` are required; an optional `affected` logical column is used as the default source for `affected` (see below). `sire`/`dam` are `NA` for a founder, and may be `NA` for only one of the two (partial parentage) – see the non-informative-trim tier above for how that case is handled.
`genotyped`	a logical vector, the same length as `nrow(ped)` and in the same row order, `TRUE` where a genotype (or other available biological sample) exists for that individual. `NA` is not allowed, matching kinship2's own `avail` validation.
`affected`	`NULL` (default) or a logical vector the same length as `nrow(ped)`. When `NULL`, defaults to `ped$affected` if that column exists, or to all-`FALSE` (unaffected) if it does not – ensuring the affected-priority trim can always make progress even with no recorded affected status.
`maxBits`	numeric, default `16L`. The bit-size budget the affected-priority trim reduces toward.

Details

A fully genotyped pedigree can shrink to zero rows (see the example).

Value

A list:

ped The shrunk pedigree `data.frame`, with all of `ped`'s original columns; a promoted founder's `sire`/ `dam` are set to `NA`.

idTrimmed Character vector, every `id` removed, in removal order.

idList A list with elements `unavail`, `noninform` and `affected` – character vectors (`character(0)` when empty) grouping `idTrimmed` by which tier removed each id.

bitSize Numeric vector: the pedigree's bit size before any trimming, after tiers 1-2, then one further value per affected- priority round.

genotyped The final `genotyped` vector, aligned to `ped$id`.

pedSizeOriginal, pedSizeIntermed, pedSizeFinal Integer row counts: original, after tiers 1-2, and final.

References

Sinnwell JP, Therneau TM, Schaid DJ (2014). "The kinship2 R Package for Pedigree Data." *Human Heredity*, 78(2), 91-93.

https://cran.r-project.org/package=kinship2

See Also

`isFounder`, `getIdsWithOneParent`

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::examplePedigree[, c("id", "sire", "dam")]
genotyped <- rep(TRUE, nrow(ped))
result <- shrinkPedigree(ped, genotyped, maxBits = 16L)
nrow(ped)
## With every individual genotyped, the whole pedigree can be trimmed away.
nrow(result$ped)
```

---

smallPed *Hypothetical 17-animal pedigree*

Description

A hypothetical pedigree. It has the following structure: structure(list(id = c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q"), sire = c("Q", NA, "A", "A", NA, "D", "D", "A", "A", NA, NA, "C", "A", NA, NA, "M", NA), dam = c(NA, NA, "B", "B", NA, "E", "E", "B", "J", NA, NA, "K", "N", NA, NA, "O", NA), sex = c("M", "F", "M", "M", "F", "F", "F", "M", "F", "F", "F", "F", "M", "F", "F", "F", "M"), gen = c(1, 1, 2, 2, 1, 3, 3, 2, 2, 1, 1, 2, 1, 1, 2, 3, 0), population = c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE)), .Names = c("id", "sire", "dam", "sex", "gen", "population"), row.names = c(NA, -17L), class = "data.frame")

Usage

```
data(smallPed)
```

Format

An object of class `data.frame` with 17 rows and 6 columns.

smallPedTree *Pedigree tree built from smallPed*

Description

A pedigree tree made from `smallPed`. It is a named list with one element per animal (17 elements, named by animal ID). Each element is a list with the components `sire` and `dam`, which hold the IDs of the animal's parents (`NA` when a parent is unknown).

Usage

```
data(smallPedTree)
```

Format

An object of class `list` of length 17.

Details

Access it using the following commands.

Examples

```
library(nprcgenekeepr)
data("smallPedTree")
```

speciesGestation *Per-species reproductive parameters*

Description

A lookup table mapping a species name to reproductive parameters used across the package. It keys the gestation window in `getPotentialParents` through `getSpeciesGestation` and the minimum breeding ages in the Genetic Value Analysis unknown-parent mean-kinship correction through `getSpeciesMinBreedingAge`. Species names are matched case- and whitespace-insensitively; any species not present falls back to 210 days for gestation and 2 years for the breeding ages. Rhesus gestation is 210 days (the historical conservative bound; typical rhesus gestation is about 165 days, per Vinson & Raboin 2015), and rhesus minimum breeding ages are male = 4, female = 2.5. The table is populated for the common colony NHP species, with gestation values as conservative upper bounds. A user can override the values through a species overrides file named in the site configuration file; `loadSpeciesOverrides` merges that file onto this table. To change the bundled table itself, edit `data-raw/speciesGestation.R` and re-run that script.

species – character species name (e.g. "RHESUS").

gestation – integer maximum gestation period in days (a conservative upper bound).

minMaleBreedingAge – numeric minimum age in years at which a male of the species can sire offspring.

minFemaleBreedingAge – numeric minimum age in years at which a female of the species can bear offspring.

Usage

```
data(speciesGestation)
```

Format

An object of class `data.frame` with 14 rows and 4 columns.

Examples

```
library(nprcgenekeepr)
data("speciesGestation")
speciesGestation
```

`summarizeKinshipValues` *Summarize simulated kinship values*

Description

Makes a data.frame object containing simulated kinship summary statistics using the counts of kinship values list from `countKinshipValues`.

Usage

```
summarizeKinshipValues(countedKValues)
```

Arguments

`countedKValues`	list object from countKinshipValues function that contains the lists `kIds`, `kValues`, and `kCounts` and no other elements.

Value

a data.frame with one row of summary statistics for each pair of IDs. Pairs whose simulated kinship values include `NA` are left out, and a data.frame with no rows and no columns is returned when every pair is left out. The columns are as follows: `id_1`, `id_2`, `min`, `secondQuartile`, `mean`, `median`, `thirdQuartile`, `max`, and `sd`.

The five-number-summary columns are taken from `fivenum`: `secondQuartile` is the lower hinge (`fivenum()[2]`, approximately the first quartile) and `thirdQuartile` is the upper hinge (`fivenum()[4]`, approximately the third quartile).

Examples

```
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_4 <- list(
  id = "J",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_5 <- list(
  id = "K",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_6 <- list(
  id = "N",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
allSimParents <- list(
  simParent_1, simParent_2, simParent_3,
  simParent_4, simParent_5, simParent_6
)

extractKValue <- function(kValue, id1, id2, simulation) {
  kValue[
    kValue$id_1 == id1 & kValue$id_2 == id2,
    paste0("sim_", simulation)
  ]
}

n <- 10
simKinships <- createSimKinships(ped, allSimParents, pop = ped$id, n = n)
kValues <- kinshipMatricesToKValues(simKinships)
extractKValue(kValues, id1 = "A", id2 = "F", simulation = 1:n)
counts <- countKinshipValues(kValues)
stats <- summarizeKinshipValues(counts)
```

summary.nprcgenekeeprErr *Summarize a studbook quality-control error list*

Description

Summarizes either a studbook quality-control error list (class nprcgenekeeprErr) or a genetic value report (class nprcgenekeeprGV); this page covers both methods.

Usage

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

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

Arguments

object	object of class nprcgenekeeprErr or nprcgenekeeprGV
...	currently unused by either method

Value

For an error list, an object of class summary.nprcgenekeeprErr. It is a list with the elements txt (a character string describing the problems found) and sp (the data frame of suspicious parents).

For a genetic value report, a character vector of class summary.nprcgenekeeprGV

Examples

```
errorList <- qcStudbook(nprcgenekeepr::pedOne,
  minSireAge = 0,
  minDamAge = 0,
  reportChanges = TRUE,
  reportErrors = TRUE
)
summary(errorList)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2L,
  minDamAge = 2L,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
```

```
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped,
  removeUninformative = FALSE,
  addBackParents = FALSE
)
geneticValue <- reportGV(ped,
  guIter = 50L, # should be >= 1000L
  guThresh = 3L,
  byID = TRUE,
  updateProgress = NULL
)
trimmedGeneticValue <- reportGV(trimmedPed,
  guIter = 50L, # should be >= 1000L
  guThresh = 3L,
  byID = TRUE,
  updateProgress = NULL
)
summary(geneticValue)
summary(trimmedGeneticValue)
```

`toCharacter` *Force dataframe columns to character*

Description

Converts designated columns of a dataframe to character. Defaults to converting columns `id`, `sire`, and `dam`.

Usage

```
toCharacter(df, headers = c("id", "sire", "dam"))
```

Arguments

`df`	a dataframe. Columns named in `headers` are converted to character.
`headers`	character vector with the names of the columns to be converted to character class. Names that are not columns of `df` are skipped silently. Defaults to `c("id", "sire", "dam")`.

Value

A dataframe with the specified columns converted to class "character" for display with xtables (in shiny)

Examples

```
library(nprcgenekeepr)
pedGood <- nprcgenekeepr::pedGood
names(pedGood) <- c("id", "sire", "dam", "sex", "birth")
class(pedGood[["id"]])
pedGood <- toCharacter(pedGood)
class(pedGood[["id"]])
```

`trimPedigree` *Trim a pedigree to a group's ancestors*

Description

Filters a pedigree down to the provided group (the probands) and all of their ancestors, removing unnecessary individuals from the studbook. By default only that filtering is done. Uninformative founders are removed only when `removeUninformative = TRUE`, and single parents are added back only when both `removeUninformative` and `addBackParents` are `TRUE`.

Usage

```
trimPedigree(
  probands,
  ped,
  removeUninformative = FALSE,
  addBackParents = FALSE
)
```

Arguments

`probands`	a character vector with the list of animals whose ancestors should be included in the final pedigree.
`ped`	datatable that is the `Pedigree`. It contains pedigree information. The fields `id`, `sire` and `dam` are required.
`removeUninformative`	logical defaults to `FALSE`. If set to `TRUE`, uninformative founders are removed. Founders (having unknown sire and dam) that appear only one time in a pedigree are uninformative and can be removed from a pedigree without loss of information.
`addBackParents`	logical defaults to `FALSE`. If set to `TRUE`, the function adds back single parents to the `p` dataframe when one parent is known. It is ignored unless `removeUninformative = TRUE`. The function `addBackSecondParents` uses the `ped` dataframe, which has full complement of parents and the `p` dataframe, which has all uninformative parents removed to add back single parents to the `p` dataframe.

Value

A pedigree containing the probands and all of their ancestors. Uninformative founders are removed only when `removeUninformative` is `TRUE`, and single parents are added back only when `addBackParents` is also `TRUE`.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minSireAge = 2,
  minDamAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
breederPed <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
trimmedPedInformative <- trimPedigree(focalAnimals, breederPed,
  removeUninformative = TRUE
)
nrow(breederPed)
nrow(trimmedPed)
nrow(trimmedPedInformative)
```

`withinIntegerRange` *Get integer within a range*

Description

Assures that what is returned is an integer within the specified range. Real values are truncated. A number too large or too small to be an integer is forced to the maximum or the minimum. Non-numerics are forced to minimum without warning. Character values that look like numbers are converted. The limits themselves are not coerced, so non-integer limits can be returned as given. If `minimum` is greater than `maximum`, `maximum` wins.

Usage

```
withinIntegerRange(int = 0L, minimum = 0L, maximum = 0L, na = "min")
```

Arguments

`int`	value to be forced within a range
`minimum`	minimum integer value.
`maximum`	maximum integer value

na if "min" then non-numerics are forced to the minimum in the range If "max" then non-numerics are forced to the maximum in the range. If not either "min" or "max" it is forced to "min".

Value

A vector of values forced to be within the specified range. These are integers unless a limit is not an integer.

Examples

```
library(nprcgenekeepr)
withinIntegerRange()
withinIntegerRange(, 0, 10)
withinIntegerRange(NA, 0, 10, na = "max")
withinIntegerRange(, 0, 10, na = "max") # no argument is not NA
withinIntegerRange(LETTERS, 0, 10)
withinIntegerRange(2.6, 1, 5)
withinIntegerRange(2.6, 0, 2)
withinIntegerRange(c(0, 2.6, -1), 0, 2)
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2)
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2, na = "max")
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2, na = "min")
```

Index

∗ **Shiny modules**
modBreedingGroupsServer, 188
modBreedingGroupsUI, 191
modCrossCenterIdentityServer, 192
modCrossCenterIdentityUI, 193
modDeidentifiedExportServer, 194
modDeidentifiedExportUI, 195
modGeneticDiversityServer, 196
modGeneticDiversityUI, 197
modGeneticValueServer, 198
modGeneticValueUI, 199
modGvAndBgDescServer, 200
modGvAndBgDescUI, 201
modInputServer, 202
modInputUI, 203
modMarkerGeneticsServer, 204
modMarkerGeneticsUI, 206
modMatePairServer, 207
modMatePairUI, 210
modORIPReportingServer, 210
modORIPReportingUI, 211
modPedigreeServer, 212
modPedigreeUI, 214
modPotentialParentsServer, 215
modPotentialParentsUI, 216
modPyramidServer, 217
modPyramidUI, 218
modSnapshotTrendsServer, 218
modSnapshotTrendsUI, 220
modSummaryStatsServer, 221
modSummaryStatsUI, 223
∗ **datasets**
exampleNprcgenekeeprConfig, 84
examplePedigree, 84
finalRpt, 91
focalAnimals, 95
lacy1989Ped, 153
lacy1989PedAlleles, 154
ped1Alleles, 234
pedDuplicateIds, 235
pedFemaleSireMaleDam, 235
pedGood, 236
pedInvalidDates, 236
pedMissingBirth, 237
pedOne, 237
pedSameMaleIsSireAndDam, 238
pedSix, 238
pedWithGenotype, 239
pedWithGenotypeReport, 239
qcBreeders, 244
qcPed, 245
qcPedGvReport, 246
rhesusGenotypes, 269
rhesusPedigree, 270
smallPed, 285
smallPedTree, 286
speciesGestation, 286
∗ **direct relatives**
getFileDirectRelatives, 107
getLkDirectAncestors, 118
getLkDirectRelatives, 118
getPedDirectRelatives, 121
∗ **genetic value analysis**
calcA, 24
calcFE, 25
calcFEFG, 27
calcFG, 28
calcFGSE, 30
calcGeneDiversity, 31
calcGU, 32
calcGUSE, 34
calcKurtosis, 35
calcNeSexRatio, 36
calcNeVariance, 37
calcRetention, 39
calcSkewness, 40
∗ **obfuscation**
mapIdsToObfuscated, 169

obfuscateDate, 224
obfuscateGenomicROH, 225
obfuscateGenotypeMatrix, 226
obfuscateId, 227
obfuscateLdBlocks, 228
obfuscateMhcHaplotypes, 229
obfuscatePed, 230
obfuscateTwinRelations, 232

addAnimalsWithNoRelative, 8
addBackSecondParents, 9
addGenotype, 10, *47*, *51*
addIdRecords, 11
addParents, 12
addSexAndAgeToGroup, 13
addUIds, 13, *100, 101*, *277, 278*
alleleFreq, 14
appendColonySnapshot, 15, *22*, *75*, *220*
applyKinshipOverrides, 16, *48, 49*, *141*, *189*, *221*, *262*
appServer, 17, *155*
appUI, *18*, 18, *211, 212*
assignAlleles, 19

backfillColonySnapshots, 20, *76*, *219, 220*, *240–242*
buildMarkerGenotypeMatrix, 22, *49, 50*, *52*, *57*, *62, 63*, *170, 171*, *173–180*, *182*, *204*, *226*

calcA, 24, *35*, *141, 142*
calcA(), *26, 27*, *29*, *31–33*, *35–39*, *41*
calcAge, 25
calcFE, 25, *36*, *38*, *159*
calcFE(), *24*, *27*, *29*, *31–33*, *35–39*, *41*
calcFEFG, 27, *31*, *263*
calcFEFG(), *24*, *26*, *29*, *31–33*, *35–39*, *41*
calcFG, 28, *30–32*, *36*, *38*, *159*
calcFG(), *24*, *26, 27*, *31–33*, *35–39*, *41*
calcFGSE, *27*, *29*, 30, *263*
calcFGSE(), *24*, *26, 27*, *29*, *32, 33*, *35–39*, *41*
calcGeneDiversity, 31, *36–38*, *263*
calcGeneDiversity(), *24*, *26, 27*, *29*, *31*, *33*, *35–39*, *41*
calcGU, 32, *34, 35*, *141, 142*, *263*
calcGU(), *24*, *26, 27*, *29*, *31, 32*, *35–39*, *41*
calcGUSE, *31*, 34, *142*, *263*
calcGUSE(), *24*, *26, 27*, *29*, *31–33*, *36–39*, *41*
calcKurtosis, 35, *41*, *75*
calcKurtosis(), *24*, *26, 27*, *29*, *31–33*, *35*, *37–39*, *41*
calcNeSexRatio, *21, 22*, 36, *38*, *263*
calcNeSexRatio(), *24*, *26, 27*, *29*, *31–33*, *35, 36*, *38, 39*, *41*
calcNeVariance, 37, *263*
calcNeVariance(), *24*, *26, 27*, *29*, *31–33*, *35–37*, *39*, *41*
calcRetention, *30, 31*, 39
calcRetention(), *24*, *26, 27*, *29*, *31–33*, *35–38*, *41*
calcSkewness, *36*, 40, *75*
calcSkewness(), *24*, *26, 27*, *29*, *31–33*, *35–39*
calcSnapshotDeltas, 41, *219, 220*
calculateSexRatio, 42
checkAncestryRules, 43, *138*, *190*, *208*, *251, 252*, *260*, *265, 266*
checkChangedColsLst, 44, *47*, *244*, *282*
checkCrossCenterMapping, 45, *192, 193*, *269*
checkErrorLst, 47, *244*
checkGenotypeFile, 47, *48*, *51*, *53*, *114*, *202*
checkKinshipOverrides, *16*, *43*, *45*, 48, *58, 59*, *252*
checkLinkageMarkerGenotypeFile, 49, *53*, *57*, *205*
checkLocusMetadata, 50, *56, 57*, *62, 63*, *175, 176*, *205, 206*
checkMarkerGenotypeFile, *23*, *49–51*, 51, *53*, *56, 57*, *172–174*, *179, 180*, *204*
checkMhcHaplotypeFile, 53, *185–188*, *205*
checkParentAge, 54
checkRequiredCols, 55
checkSequenceGenotypeFile, *53*, 56, *63*, *204, 205*
checkSnapshotHistory, *16*, *20–22*, *41*, 58, *76*, *219, 220*, *240*, *253*
checkTwinRelations, *21*, *45*, 59, *148*, *164, 165*, *189*, *222*, *232*, *254*
chooseAlleles, 60
chooseDate, 61
computeGenomicROH, 62, *205, 206*, *225*
convertAncestry, *44*, 64
convertDate, 64
convertFromCenter, 66
convertRelationships, 67, *182, 183*, *223*
convertSexCodes, 68
convertStatusCodes, 69

correctParentSex, 70
countFirstOrder, 72, *223*
countKinshipValues, 73
countLoops, 74
create_wkbk, 79
createColonySnapshot, *20–22*, 75, *219, 220*
createExampleFiles, 77
createPedTree, 77
createSimKinships, 78
cumulateSimKinships, 80

dataframe2string, 82
drawPedigreeDiagram, 83

exampleNprcgenekeeprConfig, 84
examplePedigree, 84

fillGroupMembersWithSexRatio, 85
filterKinMatrix, 87, *265, 266*
filterPairs, 88, *265*
filterReport, 89
filterThreshold, 90
finalRpt, 91
findGeneration, 91, *164*, *213*
findLoops, 92
findOffspring, 93, *119, 120*
findPedigreeNumber, 94, *213*
fivenum, *287*
fixColumnNames, 95
focalAnimals, 95

geneDrop, *51*, 96, *200*
geom_tile, *160*
get_and_or_list, 135
get_elapsed_time_str, 135
getAncestors, 98
getAnimalsWithHighKinship, 99
getAutoIdFormat, *14*, 100, *255*, *277, 278*
getBoxWhiskerDescription, 101
getChangedColsTab, 102
getConfigFileName, 102, *131*, *155*
getCurrentAge, 103
getDatedFilename, 103
getDateErrorsAndConvertDatesInPed, 104
getDemographics, 105, *279*
getDescendantPedigree, 106, *119, 120*
getEmptyErrorLst, *95*, 106
getErrorTab, 107
getFileDirectRelatives, 107, *110*, *269*
getFileDirectRelatives(), *118, 119*, *121*
getFocalAnimalPed, 109, *110*
getFocalAnimalPedFromFile, 110
getFounders, 111, *145*, *175*
getGeneticDiversityStats, 112, *196*
getGenotypes, 114, *252*
getGVGenotype, 114
getGVPopulation, 115
getIdsWithOneParent, 116, *284, 285*
getIncludeColumns, 117, *131*
getLkDirectAncestors, 118
getLkDirectAncestors(), *108*, *119*, *121*
getLkDirectRelatives, *107*, 118
getLkDirectRelatives(), *108*, *118*, *121*
getOffspring, 119
getParents, 120
getPedDirectRelatives, *107*, *119, 120*, 121
getPedDirectRelatives(), *108*, *118, 119*
getPedigree, *107, 108*, *110*, 122
getPedMaxAge, 122
getPossibleCols, *117*, 123, *131*
getPotentialParents, *85*, 125, *131*, *180*, *182*, *204*, *215–217*, *286*
getPotentialSires, 126
getProbandPedigree, *106*, *119, 120*, 127
getPyramidAgeDist, 128
getPyramidPlot, 129
getRequiredCols, *123*, 130, *131*
getSiteInfo, *18*, *105*, 130, *154, 155*, *211*, *279*, *282, 283*
getSpeciesGestation, *125, 126*, 131, *156*, *215*, *286*
getSpeciesMinBreedingAge, *54*, *125*, 132, *156*, *247*, *273*, *286*
getTokenList, 133
getVersion, 134
ggsave, *277*
groupAddAssign, 136, *190–192*, *260*
gvaConvergence, 140

hasBothParents, 143
hasGenotype, 143
headerDisplayNames, 144

is_valid_date_str, 146
isFounder, *111*, 145, *285*

kinMatrix2LongForm, 146, *252*, *265*

kinship, *17*, *21*, *79*, *81*, *97*, *112*, *141*, 147, *165*, *173, 174*, *182, 183*, *189*, *191*, *222, 223*, *262*, *265*
kinshipMatricesToKValues, *73*, 149
kinshipMatrixToKValues, 151

labkey.setDefaults, *280*
lacy1989Ped, 153
lacy1989PedAlleles, 154
loadSiteConfig, *18*, 154, *155, 156*
loadSpeciesOverrides, *132*, 155, *198*, *286*
logModuleEvent, 156, *275*

makeCEPH, 157
makeExamplePedigreeFile, 158
makeFounderStatsTable, 159, *161*
makeGeneticDiversityHeatmap, *113*, 160, *196*
makeGeneticSummaryTable, *159*, 161
makeGroupMembers, 162, *163*
makeGroupNum, *86*, *163*, 163, *164*
makeGrpNum, 163
makePedigreeDiagramData, 164, *165*
makePedigreeMatingLayout, *83*, 165
makeRelationClassesTable, 167, *223*
makeSimPed, 168
mapIdsToObfuscated, 169
mapIdsToObfuscated(), *224–226*, *228–232*
markerExpectedHeterozygosity, 170, *176, 177*, *181*, *204*
markerFst, *63*, 171, *175*, *204*, *206*
markerKinship, *23*, *49*, *51, 52*, *170*, *172, 173*, 173, *177–179*, *204*, *265*
markerLdBlock, 175, *205, 206*, *228*
markerObservedHeterozygosity, *59*, *170*, 176, *204*
markerParentageExclusion, *172*, 177, *179–182*, *204, 205*
markerParentageLikelihood, 179, *204*, *206*
markerRealizedRelatednessVariance, *175, 176*, 182, *205, 206*
meanKinship, 184
mhcHaplotypeCarriers, 185, *187, 188*, *205, 206*, *229, 230*
mhcHaplotypeFrequency, *185, 186*, 186, *205, 206*
modBreedingGroupsServer, 188, *192*, *196*, *199*, *207, 208*
modBreedingGroupsServer(), *192–195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modBreedingGroupsUI, *191*, 191, *201*
modBreedingGroupsUI(), *191*, *193–195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modCrossCenterIdentityServer, 192, *193, 194*
modCrossCenterIdentityServer(), *191–195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modCrossCenterIdentityUI, *193*, 193, *195*
modCrossCenterIdentityUI(), *191–195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modDeidentifiedExportServer, 194, *195*, *205*
modDeidentifiedExportServer(), *191–193*, *195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modDeidentifiedExportUI, *194*, 195
modDeidentifiedExportUI(), *191–194*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modGeneticDiversityServer, 196, *198*
modGeneticDiversityServer(), *191–195*, *198–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modGeneticDiversityUI, *197*, 197
modGeneticDiversityUI(), *191–195*, *197*, *199–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modGeneticValueServer, *18*, *189*, *191*, 198, *200*, *219*
modGeneticValueServer(), *191–195*, *197, 198*, *200, 201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modGeneticValueUI, *199*, 199, *201*
modGeneticValueUI(), *191–195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modGvAndBgDescServer, 200, *201*
modGvAndBgDescServer(), *191–195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modGvAndBgDescUI, *200*, 201
modGvAndBgDescUI(), *191–195*, *197–200*,

203, *206, 207*, *209–212*, *214*, *216–220*, *223*
modInputServer, *18*, 202, *203*, *274*
modInputServer(), *191–195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modInputUI, *203*, 203
modInputUI(), *191–195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–220*, *223*
modMarkerGeneticsServer, 204, *207–209*
modMarkerGeneticsServer(), *191–195*, *197–201*, *203*, *207*, *209–212*, *214*, *216–220*, *223*
modMarkerGeneticsUI, *206*, 206
modMarkerGeneticsUI(), *191–195*, *197–201*, *203*, *206*, *209–212*, *214*, *216–220*, *223*
modMatePairServer, *190*, 207, *210*
modMatePairServer(), *191–193*, *195*, *197–201*, *203*, *206, 207*, *210–212*, *214*, *216–220*, *223*
modMatePairUI, *209*, 210
modMatePairUI(), *191–193*, *195*, *197–201*, *203*, *206, 207*, *209*, *211, 212*, *214*, *216–220*, *223*
modORIPReportingServer, 210, *212*
modORIPReportingServer(), *191–193*, *195*, *197–201*, *203*, *206, 207*, *209, 210*, *212*, *214*, *216–220*, *223*
modORIPReportingUI, *211*, 211
modORIPReportingUI(), *191–193*, *195*, *197–201*, *203*, *206, 207*, *209–211*, *214*, *216–218*, *220*, *223*
modPedigreeServer, *18*, *203*, 212, *214*
modPedigreeServer(), *191–193*, *195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–218*, *220*, *223*
modPedigreeUI, *203*, *213*, 214
modPedigreeUI(), *191–193*, *195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *216–218*, *220*, *223*
modPotentialParentsServer, 215, *217*
modPotentialParentsServer(), *191–193*, *195*, *197–201*, *203*, *206, 207*, *209–212*, *214*, *217, 218*, *220*, *223*
modPotentialParentsUI, *216*, 216
modPotentialParentsUI(), *191–193*, *195*, *197–201*, *203, 204*, *206, 207*, *209–212*, *214*, *216–218*, *220*, *223*
modPyramidServer, 217, *218*
modPyramidServer(), *191–193*, *195*, *197–201*, *203, 204*, *206, 207*, *209–212*, *214*, *216–218*, *220*, *223*
modPyramidUI, *217*, 218
modPyramidUI(), *191–193*, *195*, *197–201*, *203, 204*, *206, 207*, *209–212*, *214*, *216, 217*, *220*, *223*
modSnapshotTrendsServer, *199*, 218, *220*
modSnapshotTrendsServer(), *191–193*, *195*, *197–201*, *203, 204*, *206, 207*, *209–212*, *214*, *216–218*, *220*, *223*
modSnapshotTrendsUI, *219*, 220
modSnapshotTrendsUI(), *191–193*, *195*, *197–201*, *203, 204*, *206, 207*, *209–212*, *214*, *216–218*, *220*, *223*
modSummaryStatsServer, *101*, 221, *223*
modSummaryStatsServer(), *191–193*, *195*, *197–201*, *203, 204*, *206, 207*, *209–212*, *214*, *216–218*, *220*, *223*
modSummaryStatsUI, *223*, 223
modSummaryStatsUI(), *191–193*, *195*, *197–201*, *203, 204*, *206, 207*, *209–212*, *214*, *216–218*, *220*, *223*

obfuscateDate, 224, *231*
obfuscateDate(), *169*, *225, 226*, *228–232*
obfuscateGenomicROH, 225
obfuscateGenomicROH(), *169*, *224*, *226*, *228–232*
obfuscateGenotypeMatrix, *225*, 226
obfuscateGenotypeMatrix(), *169*, *224, 225*, *228–232*
obfuscateId, 227, *231*
obfuscateId(), *169*, *224–226*, *229–232*
obfuscateLdBlocks, *176*, *205, 206*, *226*, 228
obfuscateLdBlocks(), *169*, *224–226*, *228*, *230–232*
obfuscateMhcHaplotypes, *205*, 229
obfuscateMhcHaplotypes(), *169*, *224–226*, *228, 229*, *231, 232*
obfuscatePed, *124*, *194, 195*, *225, 226*, *228–230*, 230, *232*
obfuscatePed(), *169*, *224–226*, *228–230*, *232*
obfuscateTwinRelations, *225, 226*, *228*, 232

obfuscateTwinRelations(), *169*, *224–226*, *228–231*
offspringCounts, 233

ped1Alleles, 234
pedDuplicateIds, 235
pedFemaleSireMaleDam, 235
pedGood, 236
pedInvalidDates, 236
pedMissingBirth, 237
pedOne, 237
pedSameMaleIsSireAndDam, 238
pedSix, 238
pedWithGenotype, 239
pedWithGenotypeReport, 239
plotSnapshotTrends, *220*, 240
previewBackfillSupport, *219, 220*, 241
print.summary.nprcgenekeeprErr, 242
print.summary.nprcgenekeeprGV *(*print.summary.nprcgenekeeprErr*)*, 242
processQcStudbookResult, 243, *274*

qcBreeders, 244
qcPed, 245
qcPedGvReport, 246
qcStudbook, *14*, *20*, *97*, *122*, *227*, *241*, *244*, 246, *255*, *260*, *269*, *274*

rankSubjects, 250
readAncestryRules, 251
readKinshipOverrides, *251*, 252, *253, 254*
readSnapshotHistory, *58*, *220*, 253
readTwinRelations, 254
removeAutoGenIds, *227*, *250*, 255, *277*
removeDuplicates, *97*, 255
removeEarlyDates, 256
removePotentialSires, 257
removeUninformativeFounders, 258
removeUnknownAnimals, 259
reportAncestryViolations, *138*, *190*, 260, *266*
reportGV, *17*, *21*, *27*, *31*, *33*, *35*, *58*, *75, 76*, *140–142*, *156*, *198, 199*, *250*, *252*, 261, *265*
reportMatePairs, *207–210*, 264
req, *192*, *194*, *209*, *213*
resolveCrossCenterIds, *45, 46*, *171*, *192, 193*, *204*, 268
rhesusGenotypes, *53*, *186*, *188*, 269
rhesusPedigree, 270
runGeneKeepR, *154*, 271, *272*
runModularApp, *271, 272*, 272
runQcStudbook, *244*, 273

safeExecute, *157*, 274
saveDataframesAsFiles, 275
savePlotToFile, 276
seq.Date, *21*, *241*
set_seed, *141*, 281
setAutoIdFormat, *14*, *101*, 277
setExit, 278
setLabKeyDefaults, *105*, 279
setPopulation, *76*, *213*, 280
shouldShowChangedColsTab, *18*, 282, *283*
shouldShowOripTab, *18*, *211, 212*, 282
shrinkPedigree, 283
smallPed, 285
smallPedTree, 286
speciesGestation, *126*, *131–133*, *155, 156*, *215*, *262*, 286
summarizeKinshipValues, 287
summary.nprcgenekeeprErr, 289
summary.nprcgenekeeprGV *(*summary.nprcgenekeeprErr*)*, 289

toCharacter, 290
trimPedigree, *21*, *213*, 291

withinIntegerRange, 292