Package: fishSim (via r-universe)

October 21, 2026

Title Simulate Populations With Multiple Stocks, Movement, Mating, Mortality, and Parentage

Version 0.0.0.9000

Description Simulates population demography of an arbitrary number of stocks potentially connected by movement, with full parentage record-keeping. Includes functions for movement, mating, mortality, and aging, plus convenience functions to handle data archiving.

Depends R (>= 3.4.4)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Imports doParallel, foreach, ids, mvbutils, parallel, fastmatch

Suggests knitr, rmarkdown, tinytest

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-03-24 02:17:32 UTC

RemoteUrl https://github.com/SMBaylis/fishSim

RemoteRef HEAD

RemoteSha 3f0e181e9f5a8114d81d80f4227e0caa90abc528

Contents

addmtDNA 2
altMate 3
archive_dead 5
birthdays 6

bondedMate . . . 7
both . . . 9
capture . . . 9
check_growthrate . . . 10
dfify . . . 12
findRelatives . . . 12
findRelativesPar . . . 13
fishSim . . . 15
grandparents . . . 15
great.grandparents . . . 16
great2.grandparents . . . 16
great3.grandparents . . . 17
great4.grandparents . . . 17
lookAtPair . . . 18
make_archive . . . 19
make_sim_names . . . 19
makeFounders . . . 20
mate . . . 21
mort . . . 23
move . . . 24
namedRelatives . . . 26
oldmove . . . 28
parents . . . 28
PoNG . . . 29
quickin . . . 31
remove_dead . . . 32
rTruncPoisson . . . 32
sexSwitch . . . 33
xpairs . . . 33

Index **34**

`addmtDNA` *add mtDNA haplotypes to a completed simulation*

Description

Adds a column of mtDNA haplotypes to the 'indiv' data of a completed simulation, given input mtDNA haplotypes and their frequencies at the time of the founder population. mtDNA haplotypes are inherited matrilineally (each animal gets its haplotype from its mother). Mutation is not handled, so mtDNA diversity will decrease through time via genetic drift, especially in small populations.

Usage

```
addmtDNA(indiv = makeFounders(), mtDNAfreqs)
```

Arguments

`indiv`	Individual matrix, as from `makeFounders()`. Every animal in indiv must either be a founder (see `makeFounders()`), or be descended exclusively from founders.
`mtDNAfreqs`	a data.frame of mtDNA haplotypes ('haplos') in the first column, and their frequencies ('freqs') in the second.

Value

a pop-by-9 character matrix, defined in the makeFounders documentation, plus an extra column ('mtHaplo') holding each animal's mtDNA haplotype.

Examples

```
## set up a population with some founders and their offspring
batchSize = 2
mortRate = 0.5
indiv <- makeFounders(stocks = c(1))
for( y in 1:10) {
    indiv <- altMate( indiv, batchSize = batchSize, type = "flat")
    indiv <- mort( indiv, year = y, type = "flat", mortRate = mortRate)
    indiv <- birthdays(indiv)
}
## make haplotype frequencies
haplos <- paste(expand.grid(LETTERS, LETTERS)[,1],expand.grid(LETTERS, LETTERS)[,2], sep = "")
counts <- ceiling(rexp(length(haplos), rate = 0.2))
freqs <- counts / sum(counts)
mtDNAfreqs <- data.frame(haplos = haplos, freqs = freqs)
## give all animals mtDNA haplotypes consistent with their descent
indiv_haplos <- addmtDNA(indiv, mtDNAfreqs)
```

`altMate` *Breeding based on mature females*

Description

Returns an individual matrix with added newborns at age 0. This is the second mate method, where the number of offspring is derived from the number of mature females, such that each mature female produces a number of offspring specified by a sampling distribution, and fathers are randomly drawn from all mature males within the mother's stock. Note that in this mating system, *maturity by age* is specified, rather than *fecundity by age*. A single probability distribution sets the number of offspring for each female, but the probability that an individidual female is mature may vary by age. The same maturity by age structure applies for males. It is possible, in cases where the maturity-by-age slope is shallow, that an individual may be 'mature' in one breeding season, but then 'not mature' the next season. Another mate method exists, where the number of newborns is set as a proportion of the population size, and mating occurs until the required number of offspring are generated (if possible, given breeding constraints) - see `mate()`.

Usage

```
altMate(
  indiv = makeFounders(),
  batchSize = 0.5,
  fecundityDist = "poisson",
  osr = c(0.5, 0.5),
  year = "-1",
  firstBreed = 1,
  type = "flat",
  maxClutch = Inf,
  singlePaternity = TRUE,
  exhaustFathers = FALSE,
  maturityCurve,
  maleCurve,
  femaleCurve
)
```

Arguments

indiv	A matrix of individuals, e.g., generated in `makeFounders()` or output from `move()`
batchSize	Numeric, possibly vector. The mean number of offspring produced per mature female if `fecundityDist = "poisson"`; the value of T from a zero-truncated Poisson distribution, if `fecundityDist = "truncPoisson"` (useful in cases where the probability of no offspring is rolled into the 'maturity' parameter); the probability that a female will have a (single) offspring, given that she is mature, if `fecundityDist = "binomial"`. If `fecundityDist = "multinomial"`, must be a vector of relative probability of each number of offspring, from 0 to the maximum possible number of offspring. Used for all maturity structures. Note that, if run within a loop that goes `[move -> altMate -> mort -> birthdays]`, 'produced' is not the same as 'enters age-class 1', as some individuals will die at age 0.
fecundityDist	One of "poisson", "truncPoisson", "binomial", or "multinomial". Sets the distribution of the number of offspring per mature female. Defaults to "poisson".
osr	Numeric vector with length two, `c(male, female)`, giving the sex ratio at birth (recruitment). Used to assign sexes to new offspring.
year	Intended to be used in a simulation loop - this will be the iteration number, and holds the `birthyear` value to give to new recruits.
firstBreed	Integer variable. The age at first breeding, default 1. The minimum age at which individuals can breed. Applies to potential mothers and potential fathers. `firstBreed`, `maturityCurve`, `maleCurve`, and `femaleCurve` are all capable of specifying an age at first breeding, and `firstBreed` takes precedence.
type	The type of maturity-age relationship to simulate. Must be one of `"flat"`, `"age"`, or `"ageSex"`. If `"flat"`, the probability of parenthood is the same for all age:sex combinations above firstBreed. If `"age"`, the probability that an individual is sexually mature is age-specific, set in `maturityCurve`. If `"ageSex"`,

the probability that an individual is sexually mature is age- and sex-specific, set for males in `maleCurve` and for females in `femaleCurve`.

`maxClutch` Numeric value giving the maximum clutch / litter / batch / whatever size. Reduces larger clutches to this size, for each breeding female.

`singlePaternity` Logical indicating whether all the offspring produced by female in a year should have the same father. Default `TRUE`. If `FALSE`, each offspring will have a randomly-drawn father from within the mother's stock. Note that this can lead to rapid exhaustion of fathers if `exhaustFathers = TRUE`.

`exhaustFathers` Logical indicating whether fathers should become 'exhausted' by one breeding attempt. If exhausted, an individual will only mate with one female, though may father more than one offspring - see `singlePaternity` and `batchSize`.

`maturityCurve` Numeric vector describing the age-specific probability of maturity curve. One value per age, over all ages from `0:max(indiv[,8])`. Used if `type = "age"`. Note that `firstBreed` can interfere with `maturityCurve` by setting maturities to zero for some age classes. Recommended usage is to set `firstBreed` to zero whenever `maturityCurve` is specified.

`maleCurve` Numeric vector describing age-specific probability of maturity for males. One value per age, over all ages from `0:max(indiv[,8])`. Used if `type = "ageSex"`. Note that `firstBreed` can interfere with `maleCurve` by setting maturities to zero for some age classes. Recommended usage is to set `firstBreed` to zero whenever `maleCurve` is specified.

`femaleCurve` Numeric vector describing age-specific maturity for females. One value per age, over all ages from `0:max(indiv[,8])`. Used if `type = "ageSex"`. Note that `firstBreed` can interfere with `femaleCurve` by setting maturities to zero for some age classes. Recommended usage is to set `firstBreed` to zero whenever `femaleCurve` is specified.

See Also

`mate()`

`archive_dead` *Take dead individuals and copy them to an archive*

Description

For larger simulations, the matrix `indiv` may grow very large and slow down the simulation. In these cases, run-times may be improved by periodically moving dead individuals into an archive that is read and written less frequently than `indiv`.

Usage

```
archive_dead(indiv = mort(), archive = make_archive())
```

Arguments

indiv	A matrix of individuals, as from makeFounders(), move(), mate(), or mort().
archive	A matrix of individuals, probably from make_archive() or a previous call of archive_dead().

See Also

remove_dead() make_archive()

Examples

```
archive <- make_archive()
indiv <- makeFounders()
ages <- min(indiv[,8]):200
ageMort <- 0.1 + (0.2*1/(ages+1)) ## placeholder ageMort with (extreme) negative senescence
stocks <- c(0.3,0.3,0.4)  ## matches defaults in makeFounders
admix.m <- matrix(NA, nrow = length(stocks), ncol = length(stocks))
for (i in 1:nrow(admix.m)) {
     admix.m[i,] <- stocks*stocks[i]
}
indiv <- makeFounders()
for(k in 1:30) {
   indiv <- move(indiv = indiv, moveMat = admix.m)
   indiv <- mate(indiv = indiv, osr = c(0.55,0.45), year = k)
   indiv <- mort(indiv = indiv, type = "age", ageMort = ageMort, year = k)
   indiv <- birthdays(indiv = indiv)
  archive <- archive_dead(indiv = indiv, archive = archive) # archives a copy of dead animals'
                                                            # data
   indiv <- remove_dead(indiv = indiv) # actually removes the dead from 'indiv'.
}
```

---

birthdays *Add one to each individual's age*

Description

Nothing fancy: this is separated from the other functions to allow more flexible assignments of movement, mating and mortality within each year. Only updates ages for animals that are alive, so indiv[,8] will remain 'age at death' for all dead animals.

Usage

```
birthdays(indiv = makeFounders())
```

Arguments

indiv	A matrix of individuals, as from makeFounders(), move(), mate(), or mort().

bondedMate *mating between bonded pairs*

Description

Returns a matrix with added newborns at age 0. This is the third mate method, and is similarly-structured to altMate, but with added features to handle male pair-bonds and divorce. The male pair-bond and divorce features replace single paternity and male exhaustion in `altMate`.

Usage

```
bondedMate(
  indiv = makeFounders(),
  batchSize = 0.5,
  fecundityDist = "poisson",
  osr = c(0.5, 0.5),
  year = "-1",
  firstBreed = 1,
  type = "flat",
  maxClutch = Inf,
  prDiv = 0,
  maturityCurve,
  maleCurve,
  femaleCurve
)
```

Arguments

`indiv`	A matrix of individuals, e.g., generated in `makeFounders()` or output from `move()`
`batchSize`	Numeric, possibly vector. The mean number of offspring produced per mature female if `fecundityDist = "poisson"`; the value of T from a zero-truncated Poisson distribution, if `fecundityDist = "truncPoisson"` (useful in cases where the probability of no offspring is rolled into the 'maturity' parameter); the probability that a female will have a (single) offspring, given that she is mature, if `fecundityDist = "binomial"`. If `fecundityDist = "multinomial"`, must be a vector of relative probability of each number of offspring, from 0 to the maximum possible number of offspring. Used for all maturity structures. Note that, if run within a loop that goes `[move -> altMate -> mort -> birthdays]`, 'produced' is not the same as 'enters age-class 1', as some individuals will die at age 0.
`fecundityDist`	One of "poisson", "truncPoisson", "binomial", or "multinomial". Sets the distribution of the number of offspring per mature female. Defaults to "poisson".
`osr`	Numeric vector with length two, `c(male, female)`, giving the sex ratio at birth (recruitment). Used to assign sexes to new offspring.

year	Intended to be used in a simulation loop - this will be the iteration number, and holds the birthyear value to give to new recruits.
firstBreed	Integer variable. The age at first breeding, default 1. The minimum age at which individuals can breed. Applies to potential mothers and potential fathers. firstBreed, maturityCurve, maleCurve, and femaleCurve are all capable of specifying an age at first breeding, and firstBreed takes precedence.
type	The type of maturity-age relationship to simulate. Must be one of "flat", "age", or "ageSex". If "flat", the probability of parenthood is the same for all age:sex combinations above firstBreed. If "age", the probability that an individual is sexually mature is age-specific, set in maturityCurve. If "ageSex", the probability that an individual is sexually mature is age- and sex-specific, set for males in maleCurve and for females in femaleCurve.
maxClutch	Numeric value giving the maximum clutch / litter / batch / whatever size. Reduces larger clutches to this size, for each breeding female.
prDiv	numeric between 0 and 1, giving the probability of divorce for an established pair. Divorces are assumed to occur 'before' the mating season, so both members of a divorced pair are available for mating again.
maturityCurve	Numeric vector describing the age-specific probability of maturity curve. One value per age, over all ages from 0:max(indiv[,8]). Used if type = "age". Note that firstBreed can interfere with maturityCurve by setting maturities to zero for some age classes. Recommended usage is to set firstBreed to zero whenever maturityCurve is specified.
maleCurve	Numeric vector describing age-specific probability of maturity for males. One value per age, over all ages from 0:max(indiv[,8]). Used if type = "ageSex". Note that firstBreed can interfere with maleCurve by setting maturities to zero for some age classes. Recommended usage is to set firstBreed to zero whenever maleCurve is specified.
femaleCurve	Numeric vector describing age-specific maturity for females. One value per age, over all ages from 0:max(indiv[,8]). Used if type = "ageSex". Note that firstBreed can interfere with femaleCurve by setting maturities to zero for some age classes. Recommended usage is to set firstBreed to zero whenever femaleCurve is specified.

Details

Every female will attempt to breed. If her partner is still alive and in the same stock, she will remate with that partner unless they have divorced. If her partner is not alive, has moved to another stock, or the pair has divorced, she will select a new mate from the unmated males in her stock (if any).

See Also

altMate()

Examples

```
batchSize = 2
mortRate = 0.5
```

```
indiv <- makeFounders(stocks = c(1))
for( y in 1:10) {
    indiv <- bondedMate( indiv, batchSize = batchSize, type = "flat", prDiv = 0.1)
    indiv <- mort( indiv, year = y, type = "flat", mortRate = mortRate)
    indiv <- birthdays(indiv)
}
```

---

both *Unique reals from ordered pairs of integers*

Description

Return a vector of unique real-valued numbers from a pair of integers

Usage

```
both(m1, m2)
```

Arguments

m1	a vector of integers
m2	another vector of integers

---

capture *Identify genetic captures/samples in population*

Description

Works in a manner similar to mort(), assigning a year to captured individuals and killing them if sampling is fatal. Sex specific sampling is allowed.

Usage

```
capture(
  indiv = makeFounders(),
  n = 1,
  year = "-1",
  fatal = TRUE,
  sex = NULL,
  age = NULL
)
```

Arguments

indiv	A matrix of individuals, as from makeFounders(), mate(), or mort().
n	Number of captures (genetic samples)
year	Capture year
fatal	Is sampling fatal?
sex	Sex specific sampling (either "M", "F" or NULL)
age	Integer vector. The age class(es) at which sampling occurs

---

check_growthrate *Estimate population growth under some* mate() *and* mort() *conditions*

Description

In complex models, population trends may not be immediately clear from the settings. Yet it can be important to know population trends in advance: growing populations take progressively more computing time, and the relationship dynamics between individuals across generations are different for growing vs. shrinking populations. check_growthrate() provides an estimate of the long-term population growth rate under some altMate() and mort() settings, assuming one altMate() and one mort() per cycle. Estimation is via Leslie matrices. check_growthrates only functions for some altMate() and mort() structures. Specifically, in altMate, type must be one of "flat", "age", or "ageSex", and in mort, if type is "flat" or "age", a single growth rate will be returned, but if "stock" or "ageStock", one growth rate will be returned per stock. If "type" is "simple" for mort(), the growthrate is forced to zero, so check_growthrate() does not explicitly handle this case. Some conditions can cause check_growthrate to fail or provide inaccurate estimates. If mature females go unmated through lack of available fathers (for instance, if exhaustFathers = TRUE in mate and N(mature females) > N(mature males)), the Leslie matrix approach will provide an overestimate of the growth rate. In mate(), batchSize is the mean number of offspring per female, but if maxClutch is also set to a value other than Inf, the *effective* mean number of offspring per female is estimated by simulation. The mean number of female offspring per female per year is assumed to be half of the effective mean number of offspring per female unless osr is specified, in which case the proportion of female offspring is taken from osr. If your model involves a variation not handled by check_growthrates(), you may find it simplest to run your simulation for a few generations with a smallish founder population and empirically estimate the growth rate.

Usage

```
check_growthrate(
  forceY1 = NA,
  mateType = "flat",
  mortType = "flat",
  batchSize,
  firstBreed = 1,
  maxClutch = Inf,
  osr = c(0.5, 0.5),
```

```
  maturityCurve,
  femaleCurve,
  maxAge = Inf,
  mortRate,
  ageMort,
  stockMort,
  ageStockMort
)
```

Arguments

forceY1	optionally force first-year mortality to a specific value. Defaults to NA. If non-NA, should be a numeric value between 0 and 1. If NA, ignored. Must be the first argument so that uniroot()-based solutions for null growth will work.
mateType	the value of type used in the altMate() call. Must be one of flat, age, or ageSex. If flat, batchSize must be provided. If age, maturityCurve and batchSize must be provided. If ageSex, femaleCurve and batchSize must be provided. Defaults to flat.
mortType	the value of type used in the mort() call. Must be one of flat, age, stock, or ageStock. If flat, mortRate must be provided. If age, ageMort must be provided. If stock, stockMort must be provided. If ageStock, ageStockMort must be provided. Defaults to flat.
batchSize	the value of batchSize used in the altMate() call. Cannot be blank.
firstBreed	the value of firstBreed used in the altMate() call. Defaults to 1.
maxClutch	the value of maxClutch used in the altMate() call. Defaults to Inf. If non-Inf, *effective* batchSize is estimated as the mean of 1000000 draws from the distribution of batchSize, subsetted to those <= maxAge.
osr	the value of osr used in the altMate() call. Female proportion is used as a multiplier on the fecundities. Defaults to c(0.5, 0.5).
maturityCurve	the value of maturityCurve used in the altMate() call. check_growthrates() only uses female fecundities in its estimates, so femaleCurve is equivalent to maturityCurve in check_growthrates(), but maturityCurve is used when mateType is age. If both mortality and maturity are specified as vectors, they can be of different lengths. If the maturity vector is shorter, it is padded to the same length as the mortality vector by repeating the last value in the vector.
femaleCurve	the value of femaleCurve used in the altMate() call. check_growthrates() only uses female fecundities in its estimates, so femaleCurve is equivalent to maturityCurve in check_growthrates(), but femaleCurve is used when mateType is ageSex. If both mortality and maturity are specified as vectors, they can be of different lengths. If the maturity vector is shorter, it is "padded" to the same length as the mortality vector by repeating the last value in the vector.
maxAge	the value of maxAge used in the mort() call. Defaults to Inf.
mortRate	the value of mortRate used in the mort() call
ageMort	the value of ageMort used in the mort() call. If both mortality and maturity are specified as vectors, they can be of different lengths. If the mortality vector is

shorter, it is "padded" to the same length as the maturity vector by repeating the last value in the vector.

`stockMort` the value of `stockMort` used in the `mort()` call

`ageStockMort` the value of `ageStockMort` used in the `mort()` call. If both mortality and maturity are specified as vectors, they can be of different lengths. If the mortality vector is shorter, it is "padded" to the same length as the maturity vector by repeating the last value in the vector.

See Also

`PoNG()`

---

dfify *convert an early* makeFounders*-type matrix to* data.frame

---

Description

Internal operations are all performed on matrix objects lacking human-friendly features like column names. `dfify` takes a matrix as output from `makeFounders()` and outputs a more human-readable data.frame

Usage

```
dfify(inds)
```

Arguments

`inds` A matrix of individuals, as from `mort()`

---

findRelatives *Find shared ancestors between pairs of individuals*

Description

`findRelatives` takes a set of 'sampled' individuals and a population simulation output (i.e., an `indiv` object, of the kind output by `mort()`). It returns a `data.frame` for each pair of sampled individuals, with columns:

1. `Var1`: the first individual's ID
2. `Var2`: the second individual's ID
3. `related`: `TRUE` for each pair that with one or more shared ancestors within seven ancestral generations (i.e., great-great-great-great grandparents), otherwise `FALSE`
4. `totalRelatives`: numeric indicator of the total number of shared ancestors found

5. OneTwo, OneThree, ThreeFour, etc.: Numeric values, indicating the number of shared ancestors by relationship class. For instance, a shared relative in the OneTwo class indicates that one member of the pair is the other member's parent, a shared relative in the OneThree class indicates that one member is the other's grandparent, and a shared in the ThreeFour class indicates that one individual's grandparent is the other's great- grandparent. If a pair shares an ancestor in the TwoThree class, they necessarily also share two ancestors in the ThreeFour class and four ancestors in the FourFive class, and so on. Note that relationship classes are identical by reversal - ThreeFour is the same as FourThree, and so only relationship classes with increasing order are presented (i.e., ThreeFour and OneFive are in the output, but not ThreeTwo).

Usage

```
findRelatives(indiv, sampled, delimitIndiv = TRUE)
```

Arguments

indiv	A matrix of individuals, as from mort(), but which will need to contain long strings of parent-offspring relationships, so will most likely come from a multi-generation simulation.
sampled	A character vector containing one or more IDs, e.g., from mort()[,1]
delimitIndiv	TRUE/FALSE. Lookups can be sped up markedly by first delimiting indiv to only those animals that exist as parents, or that are marked as sampled. Default TRUE.

See Also

lookAtPair()

---

findRelativesPar *Partially-parallelized* findRelatives()

Description

findRelativesPar is a partially-parallelized version of findRelatives. Its use is exactly the same, but it requires libraries foreach, parallel, and doParallel.

Usage

```
findRelativesPar(
  indiv,
  sampled = TRUE,
  verbose = TRUE,
  nCores = detectCores() - 1,
  delimitIndiv = TRUE
)
```

Arguments

indiv	A matrix of individuals, as from mort(), but which will need to contain long strings of parent-offspring relationships, so will most likely come from a multi-generation simulation.
sampled	TRUE or FALSE. If TRUE, compares only individuals marked as sampled in indiv[,9]. If FALSE, compares all individuals in indiv.
verbose	TRUE or FALSE. If TRUE, prints a table of sampling years for sampled individuals.
nCores	the number of cores to use for parallel processes. Defaults to one less than the number of cores on the machine.
delimitIndiv	TRUE/FALSE. Lookups can be sped up markedly by first delimiting indiv to only those animals that exist as parents, or that are marked as sampled. Default TRUE.

Details

findRelativesAlt uses the sample indicator in indiv,9 to decide which individuals to compare, whereas 'findRelativesPar' compares between all individuals in 'sampled'.

Lookup operations to find each member's ancestors are parallelized, but comparisons between ancestor-sets are not. On a test-set of 100 sampled individuals, this partial- parallelization reduced runtime from 55 seconds to 22 seconds.

findRelatives takes a set of 'sampled' individuals and a population simulation output (i.e., an indiv object, of the kind output by mort()). It returns a data.frame for each pair of sampled individuals, with columns:

1. Var1: the first individual's ID
2. Var2: the second individual's ID
3. related: TRUE for each pair that with one or more shared ancestors within seven ancestral generations (i.e., great-great-great-great grandparents), otherwise FALSE
4. totalRelatives: numeric indicator of the total number of shared ancestors found
5. OneTwo, OneThree, ThreeFour, etc.: Numeric values, indicating the number of shared ancestors by relationship class. For instance, a shared relative in the OneTwo class indicates that one member of the pair is the other member's parent, a shared relative in the OneThree class indicates that one member is the other's grandparent, and a shared in the ThreeFour class indicates that one individual's grandparent is the other's great- grandparent. If a pair shares an ancestor in the TwoThree class, they necessarily also share two ancestors in the ThreeFour class and four ancestors in the FourFive class, and so on. Note that relationship classes are identical by reversal - ThreeFour is the same as FourThree, and so only relationship classes with increasing order are presented (i.e., ThreeFour and OneFive are in the output, but not ThreeTwo). Note that, if there is an object called ancestors in the global environment, the foreach() loops may refer to that copy of ancestors, rather than the one generated inside findRelativesPar(). This is a known bug. Rename and delete ancestors. If it occurs, this bug will cause the following error: Error in cbind(ancestors, parents.o, grandparents.o, ggrandparents.o: n

See Also

findRelatives()

capture()

`fishSim` *fishSim*

Description

R tool for simulation of population dynamics; originally written for fish

Author(s)

Maintainer: Shane Baylis <shane.m.baylis@gmail.com>

`grandparents` *Look up the grandparents of one or more individuals*

Description

A convenience wrapper for `parents()`, returning the grandparents of one or more individuals. If FF is 'father's father', MF is 'mother's father', and so on, grandparents are returned in order FF, FM, MF, MM for each specified ID.

Usage

```
grandparents(ID, indiv)
```

Arguments

`ID`	A character vector containing one or more IDs, e.g., from `mort()[,1]`
`indiv`	A matrix of individuals, as from `mort()`.

See Also

`parents()`

great.grandparents *look up the great-grandparents of one or more individuals*

Description

A convenience wrapper for parents(), returning the great-grandparents of one or more individuals. If FFF is 'father's father's father', MFM is 'mother's father's mother', and so on, great-grandparents are returned in order FFF, FFM, FMF, FMM, MFF, MFM, MMF, MMM for each specified ID.

Usage

```
great.grandparents(ID, indiv)
```

Arguments

ID	A character vector containing one or more IDs, e.g., from mort()[,1]
indiv	A matrix of individuals, as from mort().

See Also

parents()

great2.grandparents *Look up the great-great-grandparents of one or more individuals*

Description

A convenience wrapper for parents(), returning the great-great-grandparents of one or more individuals. If 'FFFF' is 'father's father's father's father', and 'MMFM' is 'mother's mother's father's mother', and so on, great-great-grandparents are returned in order FFFF, FFFM, FFMF, FFMM, FMFF, FMFM, FMMF, FMMM, MFFF, MFFM, MFMF, MFMM, MMFF, MMFM, MMMF, MMMM

Usage

```
great2.grandparents(ID, indiv)
```

Arguments

ID	A character vector containing one or more IDs, e.g., from mort()[,1]
indiv	A matrix of individuals, as from mort().

See Also

parents()

great3.grandparents *Look up the great-great-great-grandparents of one or more individuals.*

Description

A convenience wrapper for `parents()`, returning the great-great-great-grandparents of one or more individuals. If 'FFFFF' is 'father's father's father's father's father', and 'FFMFM' is 'father's father's mother's father's mother', and so on, great-great-great-grandparents are returned in order: FFFFF, FFFFM, FFFMF, FFFMM, FFMFF, FFMFM, FFMMF, FFMMM, FMFFF, FMFFM, FMFMF, FMFMM, FMMFF, FMMFM, FMMMF, FMMMM, MFFFF, MFFFM, MFFMF, MFFMM, MFMFF, MFMFM, MFMMF, MFMMM, MMFFF, MMFFM, MMFMF, MMFMM, MMMFF, MMMFM, MMMMF, MMMMM.

Usage

```
great3.grandparents(ID, indiv)
```

Arguments

ID	A character vector containing one or more IDs, e.g., from `mort()[,1]`
indiv	A matrix of individuals, as from `mort()`.

See Also

`parents()`

great4.grandparents *Look up the great-great-great-great-grandparents of one or more individuals*

Description

A convenience wrapper for 'parents()', returning the great-great-great-great-grandparents of one or more individuals. If 'FFFFFF' is 'father's father's father's father's father's father', and 'FFFMMM' is 'father's father's father's mother's mother's mother', great-great-great-great-grandparents are returned in order FFFFFF, FFFFFM, FFFFMF, FFFFMM, FFFMFF, FFFMFM, FFFMMF, FFFMMM, FFMFFF, FFMFFM, FFMFMF, FFMFMM, FFMMFF, FFMMFM, FFMMMF, FFMMMM, FMFFFF, FMFFFM, FMFFMF, FMFFMM, FMFMFF, FMFMFM, FMFMMF, FMFMMM, FMMFFF, FMMFFM, FMMFMF, FMMFMM, FMMMFF, FMMMFM, FMMMMF, FMMMMM, MFFFFF, MFFFFM, MFFFMF, MFFFMM, MFFMFF, MFFMFM, MFFMMF, MFFMMM, MFMFFF, MFMFFM, MFMFMF, MFMFMM, MFMMFF, MFMMFM, MFMMMF, MFMMMM, MMFFFF, MMFFFM, MMFFMF, MMFFMM, MMFMFF, MMFMFM, MMFMMF, MMFMMM, MMMFFF, MMMFFM, MMMFMF, MMMFMM, MMMMFF, MMMMFM, MMMMMF, MMMMMM

Usage

```
great4.grandparents(ID, indiv)
```

Arguments

ID	A character vector containing one or more IDs, e.g., from mort()[,1]
indiv	A matrix of individuals, as from mort().

See Also

parents()

---

lookAtPair *Show a readable relationship summary for a pair of relatives*

Description

lookAtPair takes a single row from the output of findRelatives(), and returns a 7-by-7 matrix (as data.frame) showing the number of shared ancestors for each relationship class. The lookAtPair output is symmetric about the diagonal, and is particularly useful for displaying departures from expected relationship structures. For instance, the output

```
> lookAtPair(pair)
  X1 X2 X3 X4 X5 X6 X7
1  .  .  1  .  .  .  .
2  .  .  .  2  .  .  .
3  1  .  .  .  4  .  .
4  .  2  .  .  .  8  .
5  .  .  4  .  .  1 16
6  .  .  .  8  1  .  3
7  .  .  .  . 16  3  .
```

shows a pair where one individual is the other's grandparent (shown by the 1 in [1,X3], and doubling series proceeding diagonally down from that point), further related via shared great-great-grandparent / great-great-great-grandparent (the 1 in [5,X6], and doubling series proceeding diagonally from that point), and a shared great-great-great-grandparent / great-great-great-great-grandparent (the 3 in [6,X7], where we would otherwise expect a 2 from the previous shared ancestor).

Usage

```
lookAtPair(pair)
```

Arguments

pair	a data.frame object, with structure identical to a row from findRelatives().

See Also

findRelatives()

make_archive *Set up an archive matrix for storing simulation outputs*

Description

For larger simulations, the matrix `indiv` may grow very large and slow down the simulation. In these cases, run-times may be improved by periodically moving dead individuals into an archive that is read and written less frequently than `indiv`.

Usage

```
make_archive()
```

Value

The function returns an empty 0-by-8 matrix, to which subsets of `indiv` can be attached.

See Also

`archive_dead()` `remove_dead()`

make_sim_names *Make sim names, as a list*

Description

makes a list of sim names for easy extraction in `dfify`, etc.

Usage

```
make_sim_names()
```

`makeFounders` *Make a founding population*

Description

Returns a population-size-by-8 `data.frame`, with each row being an individual in the founder population.

- `[,1]` is a unique (uuid) identifier for each animal.
- `[,2]` is sex, values `"M"` or `"F"`.
- `[,3]` is `"founder"` for all animals.
- `[,4]` is `"founder"` for all animals.
- `[,5]` is the animal's birth year. Implicitly assumes that `makeFounders` occurs at the very start of year 1, just after the `birthdays` step of year 0.
- `[,6]` is `NA` for all animals. Holds the death year in most cases.
- `[,7]` is the stock membership for each animal.
- `[,8]` is the age of each animal (in 'breeding seasons') at the beginning of year 1, given that birthdays occur at the very end.
- `[,9]` is `NA` for all animals

Usage

```
makeFounders(
  pop = 1000,
  osr = c(0.5, 0.5),
  stocks = c(0.3, 0.3, 0.4),
  minAge = 1,
  maxAge = 20,
  survCurv = 0.7^(minAge:maxAge)/sum(0.7^(minAge:maxAge))
)
```

Arguments

`pop`	The size of the founder population.
`osr`	A numeric vector describing the sex ratio, `c([male], [female])`.
`stocks`	A numeric vector describing the probability that an individual is in each stock.
`maxAge`	Numeric. The max age to which an animal may survive.
`survCurv`	Numeric vector. Describes the probability within the founder cohort of belonging to each age-class for `age=-classes 1:maxAge`. Cannot be blank. Will be treated as probability weights, rather than probabilities, if the vector does not sum to 1.

Details

`makeFounders()` will throw a warning if `osr`, `stocks`, or `survCurv` do not sum to 1. It is not strictly necessary that they sum to 1 (proportionality within each class is sufficient), but error-checking and readability is easiest if they do sum to 1.

See Also

make_archive()

---

mate *Find male-female pairs and breed until a quota is filled*

Description

Returns an individual matrix with added newborns at age 0. This is the first mate method, where the number of newborns is a set as a proportion of the adult population, and matings happen between individuals that are older than the age at first-breeding (optionally with breeding success-rates set by the age of the less-mature parent, etc.,) until that 'quota' or newborns has been met, or all potential parents have reached breeding exhaustion. A second mate method exists, where the number of newborns is derived from the fecundities of all breeding females in the population - see altMate().

Usage

```
mate(
  indiv = makeFounders(),
  fecundity = 0.2,
  batchSize = 0.5,
  osr = c(0.5, 0.5),
  year = "-1",
  firstBreed = 1,
  type = "flat",
  maxClutch = Inf,
  exhaustMothers = FALSE,
  exhaustFathers = FALSE,
  fecundityCurve,
  maleCurve,
  femaleCurve
)
```

Arguments

indiv	A matrix of individuals, e.g., generated in makeFounders() or output from move()
fecundity	Numeric variable. mean number of recruits to generate, as a proportion of the number of animals in indiv. nrow(indiv)*fecundity is the annual turnover - i.e., the number of animals killed in 'mortality' will equal nrow(indiv)*fecundity in order to keep the population size constant.
batchSize	Numeric. The mean number of offspring produced per pairing. Follows a Poisson distribution. Used iff type = "flat". Note that, if run within a loop that goes [move -> mate -> mort -> birthdays], 'produced' is not the same as 'enters age-class 1', as some mortality may occur at age 0.

osr	Numeric vector with length two, c(male, female), giving the sex ratio at birth (recruitment). Used to assign sexes to new offspring.
year	Intended to be used in a simulation loop - this will be the iteration number, and holds the birthyear value to give to new recruits.
firstBreed	Integer variable. The age at first breeding, default 1. The minimum age at which individuals can breed. Applies to potential mothers and potential fathers. Both firstBreed and fecundityCurve, maleCurve, and femaleCurve are capable of specifying an age at first breeding, and firstBreed takes precedence.
type	The type of fecundity-age relationship to simulate. Must be one of "flat", "age", or "ageSex". If "flat", offspring batch sizes will be the same for all age:sex combinations. If "age", the number of offspring for each pairing is a function of fecundityCurve, with the less-fecund parent determining the batch size. If "ageSex", the number of offspring for each pairing is a function of maleCurve and femaleCurve, with the less-fecund parent determining the batch size.
maxClutch	Numeric value giving the maximum clutch / litter / batch / whatever size. Reduces larger clutches to this size, within each pairing.
exhaustMothers	Logical indicating whether mothers should become 'exhausted' by one breeding attempt. If exhausted, an individual will not mate again in this mate() call.
exhaustFathers	Logical indicating whether fathers should become 'exhausted' by one breeding attempt. If exhausted, an individual will not mate again in this mate() call.
fecundityCurve	Numeric vector dscribing the age-specific fecundity curve. One value per age, over all ages from 0:max(indiv[,8]). Used if type = "age". Note that firstBreed can interfere with fecundityCurve by setting fecundities to zero for some age classes. Recommended usage is to set firstBreed to zero whenever fecundityCurve is specified.
maleCurve	Numeric vector describing age-specific fecundity for males. One value per age, over all ages from 0:max(indiv[,8]). Used if type = "ageSex". Note that firstBreed can interfere with maleCurve by setting fecundities to zero for some age classes. Recommended usage is to set firstBreed to zero whenever maleCurve is specified.
femaleCurve	Numeric vector describing age-specific fecundity for females. One value per age, over all ages from 0:max(indiv[,8]). Used if "type = "ageSex". Note that firstBreed can interfere with femaleCurve by setting fecundities to zero for some age classes. Recommended usage is to set firstBreed to zero whenever femaleCurve is specified.

See Also

altMate()

mort *Kill some members of the population*

Description

Members are chosen according to one of several defined mortality structures. Mortality rates can bring the population to a specified size, or can be a flat probability, or a probability that depends on age, stock, or age:stock.

Usage

```
mort(
  indiv = makeFounders(),
  year = "-1",
  type = "simple",
  maxAge = if (type == "age") length(ageMort) - 1 else Inf,
  maxPop = 1000,
  mortRate,
  ageMort,
  stockMort,
  ageStockMort
)
```

Arguments

indiv	A matrix of inidviduals, as from makeFounders(), move(), or mate().
year	An integer, denoting the year in which we're killing animals
type	One of "simple", "flat", "age", "stock", or "ageStock". • If type = "simple" and the living pop > maxPop, individuals are killed at random until the living pop == maxPop. Can easily be set to never cause extinctions. • If type = "flat", individuals are killed with probability set in mortRate. Generates an exponential death curve. • If type = "age", individuals are killed with probability for their age set in ageMort. • If type = "stock", individuals are killed with probability for their stock set in stockMort. • If type = "ageStock", individuals are killed with probability for their age:stock combination set in ageStockMort.
maxAge	Sets an age above which animals *will* be killed before anything else happens. Allows a short age-specific mortality curve to be set, without checking if there are any individuals outside the range for each iteration.
maxPop	If type = "simple", the population will be reduced to this number, if not already smaller. See type.
mortRate	Numeric mortality rate. See type.

ageMort	Numeric vector of mortality rates, one for each age, ordered 0:max(age). See type.
stockMort	Numeric vector of mortality rates, one for each stock, ordered 1:max(stock). Note that stocks are numbered (as in makeFounders()), not named. Because stocks are stored as a character vector, stocks are converted via as.numeric() to associate rates with stocks. This distinction is important in cases with >9 stocks. See type.
ageStockMort	A matrix of mortality rates, with age by row and stock by column. See ageMort and stockMort for structure of rows and columns.

---

move — *Age- and/or sex-specific Markovian movement between breeding stocks*

Description

Performs movement of animals between stocks by specified probabilities of movement to-and-from each stock pair, optionally differing across age and sex. Replaces old move() - captures the same functionality and adds more, and is (much) quicker.

Usage

```
move(
  indiv = makeFounders(),
  moveMat = NULL,
  moveMat_M = NULL,
  moveMat_F = NULL
)
```

Arguments

indiv	Individual matrix, e.g. from makeFounders(). Can also be a non-founder matrix.
moveMat	A matrix describing the probability of moving from each stock to each other stock, with 'from' by row, 'to' by column, and optionally, 'age' by aisle
moveMat_M	only used if movement is sex-specific. See moveMat, but this one specifies movement for males
moveMat_F	see moveMat_M, but this one specifies movement for females

Value

a pop-by-8 character matrix, defined in the makeFounders documentation.

Examples

```
## scenario 1: movement is not age- or sex-specific:
stocks <- c(0.3,0.3,0.4)  ## matches defaults in makeFounders
admix.m <- matrix(NA, nrow = length(stocks), ncol = length(stocks))
for (i in 1:nrow(admix.m)) {
    admix.m[i,] <- stocks*stocks[i]
}  ## probability of moving into a stock is proportional to the default size of that stock
   ## from makeFounders.
indiv <- makeFounders()
indiv <- move( indiv, moveMat = admix.m)

## Scenario 2: movement is sex-specific but not age-specific.
## Make 90% of the males move to stock 3, and 90% of the females move
## to stock 1
moveMat_M <- matrix(c(rep( 0.05, 6), rep( 0.9, 3)), nrow = 3, ncol = 3)
moveMat_F <- matrix(c(rep( 0.9, 3), rep( 0.05, 6)), nrow = 3, ncol = 3)
indiv <- makeFounders()
table(indiv$Stock[ indiv$Sex == "M"]) ## where are the males before moving?
table(indiv$Stock[ indiv$Sex == "F"]) ## where are the females before moving?
indiv <- move( indiv, moveMat_M = moveMat_M, moveMat_F = moveMat_F)
table(indiv$Stock[ indiv$Sex == "M"]) ## where are the males after moving?
table(indiv$Stock[ indiv$Sex == "F"]) ## where are the females after moving?

## Scenario 3: movement is age-specific but not sex-specific. There are 6 ages (0 -> 5)
## Make animals more sedentary as they age: age 0 everyone changes stock; age 5 no-one does
moveMat <- array( data = c(0, rep(0.5,3), 0,rep(0.5,3), 0, ## age 0
                           0.1, rep(0.45,3), 0.1,rep(0.45,3), 0.1, ## age 1
                           0.2, rep(0.4,3), 0.2,rep(0.4,3), 0.2,   ## age 2
                           0.4, rep(0.3,3), 0.4, rep(0.3,3), 0.4,  ## age 3
                           0.9, rep(0.05, 3), 0.9, rep(0.05, 3), 0.9, ## age 4
                           1, rep(0,3), 1, rep(0,3), 1), ## age 5
                  dim = c(3,3,6)) ## 3 'froms', 3 'tos', and 6 ages
## this array setup is that Toby was gonna show me how to automate with formulas, I think?
indiv <- makeFounders( minAge = 0, maxAge = 5, survCurv = c(1, 0.6^(1:5)))
indiv2 <- move( indiv, moveMat = moveMat)
oldstock <- indiv$Stock
newstock <- indiv2$Stock
## what's the oldstock -> newstock pattern for 0 year-olds?
table( paste( oldstock, newstock)[ indiv$Age == 0])
any( oldstock[ indiv$Age==0] == newstock[ indiv$Age==0])
 note: for 0 year-olds, no records where oldstock == newstock

## what's the oldstock -> newstock pattern for 5 year-olds?
table( paste( oldstock, newstock)[ indiv$Age == 5])
all( oldstock[ indiv$Age==5] == newstock[ indiv$Age==5])
 note: for 5 year-olds, all records are oldstock == newstock

## Scenario 4: movement is both age- and sex-specific
## Make males sedentary except at age 1, and females randomise
## location at all ages.
moveMat_M <- array( data = c(1, rep(0,3), 1,rep(0,3), 1, ## age 0
                           rep(c(0.09,0.09,0.12), 2), 0.12, 0.12, 0.16, ## age 1
```

```
                        rep(c( 1, rep(0,3), 1,rep(0,3), 1), 4)),  ## age 2 - 5
                  dim = c(3,3,6)) ## 3 'froms', 3 'tos', and 6 ages
moveMat_F <- array( data = rep( c(rep(c(0.09,0.09,0.12), 2), 0.12, 0.12, 0.16), 6), ## all ages
                  dim = c(3,3,6)) ## 3 'froms', 3 'tos', and 6 ages
moveMat_M ## look at male moveMat
moveMat_F ## look at female moveMat
indiv <- makeFounders( minAge = 0, maxAge = 5, survCurv = c(1, 0.6^(1:5)))
indiv2 <- move( indiv, moveMat_M = moveMat_M, moveMat_F = moveMat_F)
oldstock <- indiv$Stock
newstock <- indiv2$Stock
## now, females of all ages should have moved with probability proportional
## to the starting size of the stock, and males of all ages other than 1 should
## have stayed put.
## Look at males:
table( paste( oldstock, newstock)[ indiv$Age == 0 & indiv$Sex == "M"])
## zero year-old males all stayed put
table( paste( oldstock, newstock)[ indiv$Age == 1 & indiv$Sex == "M"])
## one year-old males moved around randomly
table( paste( oldstock, newstock)[ indiv$Age == 2 & indiv$Sex == "M"])
## two year-old males all stayed put

## Look at females:
table( paste( oldstock, newstock)[ indiv$Age == 0 & indiv$Sex == "F"])
## zero year-old females moved around randomly
table( paste( oldstock, newstock)[ indiv$Age == 1 & indiv$Sex == "F"])
## one year-old females moved around randomly
table( paste( oldstock, newstock)[ indiv$Age == 2 & indiv$Sex == "F"])
## one year-old females moved around randomly
```

`namedRelatives` *Show numbers of pairs in named relationship classes*

Description

`namedRelatives` takes output from `findRelatives()` or `findRelativesPar()`, and returns counts of named relationship classes within the set of pair comparisons. Relationship classes are defined based on the closest relative shared between two individuals. So for instance, if two animals share a single parent, they will be classed as an HSP, even if there is ancestral inbreeding - for instances, in cases where the shared parent is also a half-cousin of the other parent of both individuals.

Usage

```
namedRelatives(pairs)
```

Arguments

`pairs`	a data.frame of pairwise comparisons of ancestor sets, as from `findRelatives()`

Details

The named relationship classes are:

- POPs: Parent-offspring pairs. One is the other's parent.
- GGPs: Grandparent-grandoffspring pairs. One is the other's grandparent.
- dGGPs: Double Grandparent-grandoffspring pairs. One is the other's grandparent, twice. That is, the grandparent has had offspring by two different mates, and those offspring have mated to generate the grandoffspring.
- G4Ps: Great-grandparent-great-grandoffspring pairs. One is the other's great-grandparent.
- dG4Ps: Double Great-grandparent-great-grandoffspring pairs. One is the other's great-grandparent, twice. That is, the great-grandparent has had offspring by two different mates, those offspring have produced outbred offspring, and *they* have mated to generate the great-grandoffspring.
- G6Ps: Great-great-grandparent-great-great-grandoffspring pairs. One is the other's great-great-grandparent.
- dG6Ps: Double Great-great-grandparent-great-great-grandoffspring pairs. One is the other's great-great-grandparent, twice. That is, the great-great-grandparent has had offspring by two different mates, those offspring have produced outbred offspring, and those offspring have produced outbred offspring, and *they* have mated to generate the great-great-grandoffspring.
- HSPs: Half-sibling pairs. The individuals share one parent.
- FSPs: Full-sibling pairs. The individuals share two parents.
- HTPs: Half-thiatic pairs. One individual's grandparent is the other individual's parent.
- FTPs: Full-thiatic pairs. Two of one individual's grandparents are the other individual's parents.
- HCPs: Half-cousin pairs. The individuals share one grandparent.
- FCPs: Full-cousin pairs. The individuals share two grandparents.
- GHCPs: Generalised half-cousin pairs. One individual's great-grandparent is the other individual's parent.
- GFCPs: Generalised full-cousin pairs. Two of one individual's great-grandparents are the other individual's parents.
- ORCs: Other Relationship Classes. Within the seven-generation search space, at least one shared ancestor was detected, but the relationship does not fall into one of the listed relationship classes.

See Also

`findRelatives()`

oldmove *Markovian movement between breeding stocks*

Description

Now superseded by new 'move'. New 'move' will accept the same inputs for backwards-compatability, but is faster and can handle age- and/or sex-specific movement patterns.

Usage

```
oldmove(indiv = makeFounders(), moveMat)
```

Arguments

indiv	Individual matrix, e.g. from makeFounders(). Can also be a non-founder matrix.
moveMat	An s-by-s matrix describing the probability of moving from each stock to each other stock, with 'from' by row and 'to' by column. Cannot be blank.

Value

a pop-by-8 character matrix, defined in the makeFounders documentation.

parents *Look up the parents of one or more individuals*

Description

The parents() function searches an indiv matrix for individuals with a given ID, and returns the IDs of its parents. If one or more of the parents were founders, it returns "founder" instead of a ID for founder parents, and if the individual's ID is "founder", it returns "founder" as the ID for both parents. Parents are returned in order [father, mother]. If more than one ID is given, parents are returned in order of ID: [ID1 father, ID1 mother, ID2 father, ID2 mother ...]

Usage

```
parents(ID, indiv)
```

Arguments

ID	A character vector containing one or more IDs, e.g., from mort()[,1]
indiv	A matrix of individuals, as from mort().

See Also

grandparents() great.grandparents() great4.grandparents()

Examples

```
indiv <- makeFounders()
for(k in 1:30) {
  indiv <- mate(indiv = indiv, year = k)
  indiv <- mort(indiv = indiv, year = k)
  indiv <- birthdays(indiv = indiv)
}  ## set up a population
ID <- indiv[nrow(indiv), 1] ## an individual from the youngest generation
parents(ID, indiv) ## that individual's parents, as [father, mother]
parents(parents(ID, indiv), indiv) ## that individual's grandparents, as
## [pat. grandfather, pat. grandmother, mat. grandfather, mat. grandmother]
```

PoNG *Find a Point of No Growth (by tweaking first-year mortality)*

Description

This is essentially a wrapper for `check_growthrate()`, taking all the same inputs. It returns a plot of projected growth-rates across all possible first-year survival rates and the numeric first-year survival rate at which zero population growth is expected (via uniroot, or something). The reasoning behind this utility is that often in biological systems, adult survival rates and fecundities may be quite well-characterised, and population growth rates may also be well characterised, but first-year survival may be nearly impossible to assess. This utility allows a value of first-year survival to be chosen such that the population size does not change, while leaving all well-characterised adult survival parameters unchanged. It is also useful for answering questions of the form: 'how high would our first-year survival have to be, in order for this population to *not* be in decline?', which will surely be familiar in applied management situations. Note that `PoNG()` uses some brute-force methods on the back end, so it's not terribly efficient. It takes around 5 - 10 seconds per stock on a fairly-modern (vintage 2018) laptop.

Usage

```
PoNG(
  mateType = "flat",
  mortType = "flat",
  batchSize,
  firstBreed = 1,
  maxClutch = Inf,
  osr = c(0.5, 0.5),
  maturityCurve,
  femaleCurve,
  maxAge = Inf,
  mortRate,
  ageMort,
  stockMort,
  ageStockMort
)
```

Arguments

mateType	the value of type used in the altMate() call. Must be one of flat, age, or ageSex. If flat, batchSize must be provided. If age, maturityCurve and batchSize must be provided. If ageSex, femaleCurve and batchSize must be provided. Defaults to flat.
mortType	the value of type used in the mort() call. Must be one of flat, age, stock, or ageStock. If flat, mortRate must be provided. If age, ageMort must be provided. If stock, stockMort must be provided. If ageStock, ageStockMort must be provided. Defaults to flat.
batchSize	the value of batchSize used in the altMate() call. Cannot be blank.
firstBreed	the value of firstBreed used in the altMate() call. Defaults to 1.
maxClutch	the value of maxClutch used in the altMate() call. Defaults to Inf. If non-Inf, *effective* batchSize is estimated as the mean of 1000000 draws from the distribution of batchSize, subsetted to those <= maxAge.
osr	the value of osr used in the altMate() call. Female proportion is used as a multiplier on the fecundities. Defaults to c(0.5, 0.5).
maturityCurve	the value of maturityCurve used in the altMate() call. check_growthrates() only uses female fecundities in its estimates, so femaleCurve is equivalent to maturityCurve in check_growthrates(), but maturityCurve is used when mateType is age. If both mortality and maturity are specified as vectors, they can be of different lengths. If the maturity vector is shorter, it is "padded" to the same length as the mortality vector by repeating the last value in the vector.
femaleCurve	the value of femaleCurve used in the altMate() call. check_growthrates() only uses female fecundities in its estimates, so femaleCurve is equivalent to maturityCurve in check_growthrates(), but femaleCurve is used when mateType is ageSex. If both mortality and maturity are specified as vectors, they can be of different lengths. If the maturity vector is shorter, it is "padded" to the same length as the mortality vector by repeating the last value in the vector.
maxAge	the value of maxAge used in the mort() call. Defaults to Inf.
mortRate	the value of mortRate used in the mort() call
ageMort	the value of ageMort used in the mort() call. If both mortality and maturity are specified as vectors, they can be of different lengths. If the mortality vector is shorter, it is "padded" to the same length as the maturity vector by repeating the last value in the vector.
stockMort	the value of stockMort used in the mort() call
ageStockMort	the value of ageStockMort used in the mort() call. If both mortality and maturity are specified as vectors, they can be of different lengths. If the mortality vector is shorter, it is "padded" to the same length as the maturity vector by repeating the last value in the vector.

See Also

check_growthrate()

Examples

```
batchSize = 0.8
firstBreed = 1
mortRate = 0.2
PoNG(batchSize = batchSize, firstBreed = firstBreed, mortRate = mortRate)

mortType = "stock"
stockMort = c(0.2, 0.3, 0.5)
firstBreed = 1
batchSize = 0.9
PoNG(mortType = "stock", batchSize = batchSize, firstBreed = firstBreed, stockMort = stockMort)
 ## note that only two of the stocks return a valid PoNG - with 0.5 mortality, stock 3 cannot
 ## reach null growth with any first-year survival rate between 0 and 1.
```

---

quickin *Quick lookup of CKMR-relevant relationships*

Description

`quickin` performs quick lookup of the kinships directly relevant to close-kin mark-recapture. It returns a list of eight character arrays, with each array holding one kinship in one pair of animals per row.

Usage

```
quickin(inds, max_gen = 2)
```

Arguments

`inds`	an `indiv` matrix, as from `mort()`, with some individuals marked as 'captured'
`max_gen`	the maximum depth to look up relatives, in generations. `max_gen = 2` is sufficient for relatives used in CKMR

Details

The named relationship classes (in list order) are:

- POPs: Parent-offspring pairs. One is the other's parent.
- HSPs: Half-sibling pairs. The individuals share one parent.
- FSPs: Full-sibling pairs. The individuals share two parents.
- GGPs: Grandparent-grandoffspring pairs. One is the other's grandparent.
- HTPs: Half-thiatic pairs. One individual's grandparent is the other individual's parent.
- FTPs: Full-thiatic pairs. Two of one individual's grandparents are the other individual's parents.
- HCPs: Half-cousin pairs. The individuals share one grandparent.
- FCPs: Full-cousin pairs. The individuals share two grandparents.

See Also

findRelatives()

capture()

---

remove_dead *Remove the dead from a population*

Description

For larger simulations, the matrix indiv may grow very large and slow down the simulation. In these cases, run-times may be improved by periodically moving dead individuals into an archive that is read and written less frequently than indiv. See also archive_dead().

Usage

```
remove_dead(indiv = mort())
```

Arguments

indiv	A matrix of individuals, as from mort().

See Also

archive_dead(), make_archive()

---

rTruncPoisson *Draw from a zero-truncated Poisson dist*

Description

Returns a vector of draws from a zero-truncated Poisson distribution, with specified lambda.

Usage

```
rTruncPoisson(n = 1, T = 0.5)
```

Arguments

n	The number of values to draw
T	The value of lambda for an equivalent non-truncated Poisson

sexSwitch — *Induce sex-switching in a subset of the population*

Description

Given an individual-data matrix, this function stochastically switches the sex of individuals with a set probability. Can be one-directional (i.e., only male-to-female or only female-to-male switches), or bidirectional.

Usage

```
sexSwitch(indiv = makeFounders(), direction = "both", prob = 1e-04)
```

Arguments

indiv	A matrix of individuals, as from makeFounders(), mate(), or mort().
direction	One of "MF", "FM", or "both", indicating male-to-female, female-to-male, or both directions.
prob	A numeric switching probability.

xpairs — *Look up shared ancestors for* quickin*-style kinship reporting*

Description

Targets one specific shared ancestor-type, matrix-wise between pairs of animals.

Usage

```
xpairs(anc1s, anc2s, same, weed = NULL)
```

Arguments

anc1s	a matrix of ancestors for animal 1
anc2s	a matrix of ancestors for animal 2
same	TRUE/FALSE. TRUE if you're looking for shared ancestors at the same position from each member of the pair (e.g., if you're looking for ancestors that are the mother of both pair members.
weed	if non-NULL, will attempt to remove pairs given as the parameter value.

Index

,9, *14*

addmtDNA, 2
altMate, 3, *10*
altMate(), *8*, *10, 11*, *21, 22*, *30*
archive_dead, 5
archive_dead(), *6*, *19*, *32*

birthdays, 6
bondedMate, 7
both, 9

capture, 9
capture(), *14*, *32*
check_growthrate, 10, *10*
check_growthrate(), *10*, *29, 30*
check_growthrates, *10*
check_growthrates(), *10, 11*, *30*

dfify, 12

FF, FM, MF, MM, *15*
findRelatives, 12
findRelatives(), *14*, *18*, *26, 27*, *32*
findRelativesPar, 13
findRelativesPar(), *26*
fishSim, 15
fishSim-package *(*fishSim*)*, 15

grandparents, 15
grandparents(), *28*
great.grandparents, 16
great.grandparents(), *28*
great2.grandparents, 16
great3.grandparents, 17
great4.grandparents, 17
great4.grandparents(), *28*

lookAtPair, 18
lookAtPair(), *13*

make_archive, 19
make_archive(), *6*, *21*, *32*
make_sim_names, 19
makeFounders, *3*, 20, *24*, *28*
makeFounders(), *6*, *10*, *12*, *21*, *23, 24*, *33*
mate, *10*, 21
mate(), *3*, *5, 6*, *10*, *23*, *33*
mort, *10*, 23
mort(), *6*, *9–18*, *30–33*
move, 24
move(), *6*, *21*, *23*

namedRelatives, 26

oldmove, 28

parents, 28
parents(), *15–18*
PoNG, 29
PoNG(), *12*, *29*

quickin, 31

remove_dead, 32
remove_dead(), *6*, *19*
rTruncPoisson, 32

sexSwitch, 33

uniroot, *29*
uniroot(), *11*

xpairs, 33