Package 'kindisperse'

October 28, 2026

Title Simulate and Estimate Close-Kin Dispersal Kernels

Version 0.10.3

Description Functions for simulating and estimating kinship-related dispersal. Based on the methods described in M. Jasper, T.L. Schmidt., N.W. Ahmad, S.P. Sinkins & A.A. Hoffmann (2019) <doi:10.1111/1755-0998.13043> ``A genomic approach to inferring kinship reveals limited intergenerational dispersal in the yellow fever mosquito". Assumes an additive variance model of dispersal in two dimensions, compatible with Wright's neighbourhood area. Simple and composite dispersal simulations are supplied, as well as the functions needed to estimate parent-offspring dispersal for simulated or empirical data, and to undertake sampling design for future field studies of dispersal. For ease of use an integrated Shiny app is also included.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/moshejasper/kindisperse

RoxygenNote 7.3.3

Suggests testthat, knitr, rmarkdown

Imports ggplot2, readr, shiny, shinythemes, ggrepel, LaplacesDemon, here, stats, methods, tibble, grid, magrittr, dplyr, rlang, tidyselect, stringr

VignetteBuilder knitr

Depends R (>= 2.10)

Collate 'DispersalModel.R' 'KinPairData.R' 'dispersal_model.R' 'KinPairSimulation.R' 'app.R' 'app_ports.R' 'axial_helper_functions.R' 'axials_standard.R' 'data.R' 'export_functions.R' 'import_kinpairs.R' 'kindisperse-package.R' 'sample_kindist.R' 'simgraph_data.R' 'simgraph_graph.R' 'simulate_kindist_composite.R' 'simulate_kindist_custom.R' 'simulate_kindist_simple.R'

NeedsCompilation no

Author Moshe-Elijah Jasper [aut, cre] (ORCID: <https://orcid.org/0000-0003-4541-3223>)

Maintainer Moshe-Elijah Jasper <moshe.jasper@unimelb.edu.au>

Repository CRAN

Date/Publication 2026-10-28 00:30:02 UTC

Contents

access_sigmas . . . 3
axials . . . 5
axials_add . . . 6
axials_combine . . . 7
axials_decompose . . . 7
axials_standard . . . 8
axials_subtract . . . 12
axpermute . . . 13
axpermute_standard . . . 14
axpermute_subtract . . . 18
breeding_cycle . . . 19
breeding_stage . . . 20
check_valid_kinship . . . 20
check_valid_lifestage . . . 21
csv_to_kinpair . . . 21
df_to_kinpair . . . 22
DispersalModel-class . . . 23
dispersal_model . . . 26
dispersal_vector . . . 29
display_appdata . . . 30
distances . . . 30
elongate . . . 31
filtertype . . . 32
filter_methods . . . 32
fs . . . 34
get_dispersal_model . . . 35
hs . . . 36
is.DispersalModel . . . 36
is.KinPairData . . . 37
is.KinPairSimulation . . . 37
kernelshape . . . 38
kerneltype . . . 38
KinPairData-class . . . 39
KinPairSimulation-class . . . 41
KinPairSimulation_composite . . . 44
KinPairSimulation_custom . . . 45
KinPairSimulation_simple . . . 47
kinpair_to_csv . . . 48
kinpair_to_tibble . . . 49
kinpair_to_tsv . . . 49
kinship . . . 50

kin_pair_data . . . 51
kin_pair_simulation . . . 51
lifestage . . . 53
mentari . . . 54
mount_appdata . . . 55
read_kindata . . . 56
rebase_dims . . . 56
reset_appdata . . . 57
reset_tempdata . . . 58
retrieveall_appdata . . . 58
retrieve_appdata . . . 59
retrieve_tempdata . . . 60
run_kindisperse . . . 61
sample_kindist . . . 61
sampling_stage . . . 63
simdims . . . 64
simgraph_data . . . 65
simgraph_graph . . . 66
simtype . . . 67
simulate_kindist_composite . . . 68
simulate_kindist_custom . . . 70
simulate_kindist_simple . . . 73
stages . . . 75
tsv_to_kinpair . . . 76
unmount_appdata . . . 77
vector_to_kinpair . . . 78
visible_stage . . . 78
write_kindata . . . 79

Index **80**

`access_sigmas` *Access or assign dispersal sigmas of* `KinPairSimulation` *objects*

Description

These generics & methods work with `KinPairSimulation` objects to access & modify information about the dispersal sigma parameters that define the stored simulation. The `posigma()` method accesses the single dispersal parameter stored in a simulation with `simtype == "simple"`. The remaining parameters access the dispersal parameters stored in a simulation with `simtype == "composite"`. The dispersal kernel sigma parameters of `simtype == "custom"` simulations are not yet implemented here. Assignment operations currently only exist as generics (they are not yet applied to the `KinPairSimulation` class).

Usage

```
posigma(x)

posigma(x) <- value

initsigma(x)

initsigma(x) <- value

breedsigma(x)

breedsigma(x) <- value

gravsigma(x)

gravsigma(x) <- value

ovisigma(x)

ovisigma(x) <- value

## S4 method for signature 'KinPairSimulation'
posigma(x)

## S4 method for signature 'KinPairSimulation'
initsigma(x)

## S4 method for signature 'KinPairSimulation'
breedsigma(x)

## S4 method for signature 'KinPairSimulation'
gravsigma(x)

## S4 method for signature 'KinPairSimulation'
ovisigma(x)
```

Arguments

`x`	object of class `KinPairSimulation`
`value`	new value to assign
`KinPairSimulation`	object of class KinPairSimulation

Value

`numeric` value of specified sigma parameter or modified `KinPairSimulation` object

Functions

- posigma(KinPairSimulation):
- initsigma(KinPairSimulation):
- breedsigma(KinPairSimulation):
- gravsigma(KinPairSimulation):
- ovisigma(KinPairSimulation):

See Also

Other kpsmethods: filter_methods, kernelshape(), kerneltype(), simtype()

---

axials *Estimate the axial dispersal distance of a kernel*

Description

This function performs a basic estimation of axial dispersal for a numeric vector of distances between close kin dyads. The axial dispersal distance returned is interpretable as the standard deviation of one dimension of a symmetric bivariate random distribution centred on zero.

Usage

```
axials(valvect, composite = 1)
```

Arguments

valvect	A numeric vector of distances between close kin OR an object of class KinPairData
composite	numeric. The number of separate 'draws' (dispersal events) from the kernel required to produce the final positions of the measured individuals. For example, the displacement of a child from parent at the same lifestage would involve 1 draw and thus be composite = 1. Two full siblings would be two draws (composite = 2) from the FS kernel. Non-symmetric relationships (e.g. AV, 1C) should not be decomposed using this method, nor should any assumptions be made about different kernels (e.g. the 1C relationship would appropriately be given the value 2, but not 4)

Value

Returns the value of the estimated axial dispersal distance of the kernel producing the dispersal distances measured. (numeric)

See Also

Other axial_helpers: axials_add(), axials_decompose(), axials_subtract(), axpermute(), axpermute_subtract()

Examples

```
po_dists <- c(5, 6, 7.5)
axials(po_dists) # one 'draw' (dispersal event) goes into the parent offspring category
# so composite is left to its default of 1

fs_dists <- c(2, 3, 3)
axials(fs_dists, composite = 2) # two 'draws' (symmetric dispersal events)
# go into the full sibling category so composite is set to 2
```

---

axials_add *Add axial distributions*

Description

Add axial distributions. Useful to construct an overall distribution that results from multiple 'draws' from smaller distributions. E.g. The pathway between first cousins which can be found by adding each of the component distributions of their respective lifespans along with the relevant offspring producing (e.g. oviposition) of the parent.

Usage

```
axials_add(axvals)
```

Arguments

axvals	numeric. vector of axial distribution values from different kernels that are to be added.

Value

numeric Returns the axial value that results from adding the input axial values under an additive variance framework.

See Also

Other axial_helpers: axials(), axials_decompose(), axials_subtract(), axpermute(), axpermute_subtract()

Examples

```
fullsibs_ax <- 5
parent_offspring_ax <- 25
cousin_ax <- axials_add(c(fullsibs_ax, parent_offspring_ax))
```

axials_combine *Combine axial distributions to produce a mixed distribution*

Description

combines axial distributions to produce a mixed distribution. This is useful in settings where you have two separate distributions (e.g. FS & HS) with their own axial values, but you want to average them appropriately so that they can be compared to e.g. a mixed distribution of full & half cousins which cannot be distinguished via kinship determination methods and thus are best treated as an even mixture of the two categories. Different to adding dispersal events.

Usage

```
axials_combine(axvals)
```

Arguments

axvals	numeric. vector of axial distribution values from different kernels that are to be combined

Value

numeric Returns the axial value that results from combining the input axial values under an additive variance framework.

Examples

```
fullax <- axials(c(2, 4, 5), composite = 2)
halfax <- axials(c(6, 5, 7), composite = 2)
sibax <- axials_combine(c(fullax, halfax))
```

axials_decompose *Decompose an axial distribution into simple components*

Description

Decomposes an axial distribution into simple components. Note that this should only be used in the simplest situations. It assumes all composite dispersal events are of identical magnitude and have happened equivalently to both branches of a 'symmetric' pedigree leading to the final kin dyad. (it can be used to derive e.g.full-sibling dispersal parameters from the distribution of full-siblings, or equivalent for first cousins, but **not** to divide the 'avuncular' kernel into its component parts (uncle/aunt & niece/nephew have different dispersal paths from their common ancestor)).

Usage

```
axials_decompose(ax, n_composites = 2)
```

Arguments

ax	numeric. The axial value to be decomposed.
n_composites	numeric. The number of separate 'draws' (dispersal events) from the kernel required to produce the final positions of the measured individuals. For example, the displacement of a child from parent at the same life stage would involve 1 draw and thus be composite = 1. Two full siblings would be two draws (composite = 2) from the FS kernel. Non-symmetric relationships (e.g. AV, 1C) should not be decomposed using this method, nor should any assumptions be made about different kernels (e.g. the 1C relationship would appropriately be given the value 2, but not 4)

Value

Returns the (numeric) axial distribution value of the underlying dispersal kernel from which the composite kernel was (or could be) created.

See Also

Other axial_helpers: `axials()`, `axials_add()`, `axials_subtract()`, `axpermute()`, `axpermute_subtract()`

Examples

```
fs_vect <- c(10, 11, 12)
fs_axial_raw <- axials(fs_vect, composite = 1) # composite hasn't corrected for two dispersal events
# inherent to this kin category!
fs_axial_final <- axials_decompose(fs_axial_raw, n_composites = 2)
```

---

axials_standard *Calculate the intergenerational (PO) dispersal kernel from the distributions of close kin*

Description

This function takes (at least) two vectors of kinship dispersal distances from defined kinship categories, and returns a resulting calculation of the parent-offspring (intergenerational) kinship dispersal kernel. Dispersal distances can be inputted as numeric vectors, or alternatively as objects of classes `KinPairData` or `KinPairSimulation`.

Usage

```
axials_standard(
  avect,
  bvect,
  acat = NULL,
  bcat = NULL,
  amix = FALSE,
  bmix = FALSE,
```

```
  amixcat = NULL,
  bmixcat = NULL,
  acomp = FALSE,
  bcomp = FALSE,
  acompvect = NULL,
  bcompvect = NULL,
  acompcat = NULL,
  bcompcat = NULL,
  acycle = NULL,
  bcycle = NULL,
  amixcycle = NULL,
  bmixcycle = NULL,
  acompcycle = NULL,
  bcompcycle = NULL,
  override = FALSE
)
```

Arguments

avect	vector a of kin dispersal distances for the less closely related kinship category OR object of class KinPairData.
bvect	vector b of kin dispersal distances for the more closely related kinship category OR object of class KinPairData.
acat	kinship category of kin dispersal vector avect. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
bcat	kinship category of kin dispersal vector bvect. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
amix	logical describing whether vector a is a mixture of two kinship categories. Used with amixcat. Default FALSE.
bmix	logical describing whether vector b is a mixture of two kinship categories. Used with bmixcat. Default FALSE.
amixcat	mixture kinship category of vector a. Must be set if amix == TRUE. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
bmixcat	mixture kinship category of vector b. Must be set if bmix == TRUE. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
acomp	logical denoting whether vector a should be composited with an additional kinship category vector. Used with acompvect and acompcat. Default FALSE.
bcomp	logical denoting whether vector b should be composited with an additional kinship category vector. Used with bcompvect and bcompcat. Default FALSE.
acompvect	vector acomp of kin dispersal distances for compositing with vector a OR object of class KinPairData. Must be set if acomp == TRUE.

`bcompvect` vector bcomp of kin dispersal distances for compositing with vector b OR object of class KinPairData. Must be set if bcomp == TRUE.

`acompcat` kinship category of kin dispersal vector acompvect. Must be set if acomp == TRUE. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"

`bcompcat` kinship category of kin dispersal vector bcompvect. Must be set if bcomp == TRUE. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"

`acycle` breeding cycle number of kin dispersal vector avect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`bcycle` breeding cycle number of kin dispersal vector bvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan

`amixcycle` breeding cycle number of kin dispersal vector amixvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan

`bmixcycle` breeding cycle number of kin dispersal vector bmixvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`acompcycle` breeding cycle number of kin dispersal vector acompvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`bcompcycle` breeding cycle number of kin dispersal vector bcompvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`override` whether or not to override the default -1 cycle compatibility check (default FALSE) override in situations where you are confident e.g. a c(-1, -1) cycle FS

or HS category is truly zeroed (& thus separated from later stages by a complete lifespan)

Details

This (with its paired function `axpermute_standard`) are the core functions implemented in the `kindisperse` package. They enable the decomposition of the pedigree & dispersal information contained in the sampled distributions of close kin dyads (full siblings, first cousins, etc.) & its leveraging within an additive dispersal framework to estimate the key intergenerational (parent-offspring) dispersal parameter of a population. Four key ideas underpin the approach in this function: (a) tracing dispersal pedigrees to determine the number of complete intergenerational (breeding-cycle-spanning) dispersal events separating the sampled close-kin dyads; (b) using kin categories that share the same overarching kinship 'phase' to control for residual 'phased' (non-intergenerational) disperal events that occur at the pedigree branch point (e.g. ovipositional dispersal for full sibling mosquitoes), and (c) using synced or equivalent sampling points to eliminate non-intergenerational dispersal at the branch-tips of the pedigrees, then finally (d) decomposing the 'pure' pedigree-associated (intergenerational) dispersal into an estimae of the single-generation intergenerational dispersal parameter.

At its most basic, this function requires information about two dispersal vectors, a & b - both of a phased kinship category, & `vector a` having a more dispersed pedigree than `vector b`. In addition to this initial pair of dispersed kin categories, either one or another matched pair of kin categories can be added:

1. A mixture category. This redefines the vector it is paired with (either a or b) so that rather than being considered as a 'pure' pedigree variant, it is considered as mixed with a different kin category, often of a differing pedigree phase. If used, the other initial vector must also be paired with a related mixture category or composite vector.
2. A composite dispersal vector. This is defined exactly as the initial dispersal vectors. After calculation, the axial value found is composited with that of the matched initial vector, and its kinship category redefined as a mixture category as above. If used, the other initial vector must also be paired with a related mixture category or composite vector. These can be paired so that a mixture category (e.g. first & half-first cousins where these could not be separated with available genetic data) can be counterbalanced with the composition of full sibling & half-sibling dyads, which (assuming equal mixture) approximately controls for the phasing of the mixed kin categories, enabling an estimate of intergenerational dispersal without exact knowledge of the composition of the cousins distribution.

Each vector or `KinPairData` / `KinPairSimulation` object is paired with several other parameters: (1) a logical (e.g. `amix` delineating whether the category is being used in the calculation, (2) a category parameter (.e.g `acat`) defining what kin relationship is being measured, (3) an optional breeding cycle number (e.g. `acycle`) showing the number of breeding cycles each member of the kin pair has passed through before being sampled (the cycle vector `c(1, 0)` corresponds to an adult & a juvenile being sampled at the same point in the breeding cycle; `c(1, 1)` represents two adults (i.e. after their first breeding), etc.) . If a `KinPairData` or `KinPairSimulation` object is inputted, all paired parameters that are not explicitly set will default to those contained in the objects (using KinPair objects is the ideal way to deploy this function).

For further information on this function, package & the dispersal estimation method it represents, see the paper by Jasper et al. - "A genomic approach to inferring kinship reveals limited intergenerational dispersal in the yellow fever mosquito", doi:10.1111/17550998.13043.

Value

Returns a numeric estimate of PO (intergenerational) dispersal kernel axial distribution.

See Also

Other axstandard: `axpermute_standard()`

Examples

```
cous <- rexp(100, 1 / 100)
fullsibs <- rexp(50, 1 / 50)
axials_standard(cous, fullsibs, acat = "1C", bcat = "FS")
```

`axials_subtract` *Subtract axial distributions*

Description

Subtract axial distributions, finding the difference (under an additive variance framework). This is most useful when one distribution subsumes another and includes a unique dispersal event that needs to be extracted. For example, the FS category is subsumed by the 1C category, which can be written 'FS + PO'. In this circumstance, subtracting FS from 1C will yield an estimate of the PO kernel (the basic intergenerational dispersal kernel)

Usage

```
axials_subtract(abig, asmall)
```

Arguments

`abig`	numeric. The axial dispersal distance of the larger (subsuming) distribution (e.g. 1C).
`asmall`	numeric. The axial dispersal distance of the smaller (subsumed) distribution (e.g. FS).

Value

`numeric` Returns an estimate of the axial dispersal distance of those dispersal elements that are unique to the larger dispersal distribution (e.g. PO).

See Also

Other axial_helpers: `axials()`, `axials_add()`, `axials_decompose()`, `axpermute()`, `axpermute_subtract()`

Examples

```
axials_subtract(100, 70)
```

`axpermute` *Estimate the axial dispersal distance of a kernel with confidence intervals*

Description

This function performs an estimation of axial dispersal for a numeric vector of distances between close kin dyads with confidence intervals. The axial dispersal distance returned is interpretable as the standard deviation of one dimension of a symmetric bivariate random distribution centred on zero. Confidence intervals are assigned via bootstrapping, or optionally the vector of all bootstrapped results can be outputted by setting `output` to `'vect'`, enabling its passing to other functions or external statistical analysis.

Usage

```
axpermute(vals, nreps = 1000, nsamp = "std", composite = 1, output = "confs")
```

Arguments

`vals`	numeric. Vector of distances between close kin OR object of class KinPairData.
`nreps`	numeric. Number of permutations to run for confidence intervals (default 1000)
`nsamp`	numeric. Number of kin pairs to subsample for each permutation. Either "std" or an integer. If "std" will be computed as equal to the sample size. (default "std")
`composite`	numeric. The number of separate 'draws' (dispersal events) from the kernel required to produce the final positions of the measured individuals. For example, the displacement of a child from parent at the same lifestage would involve 1 draw and thus be composite = 1. Two full siblings would be two draws (composite = 2) from the FS kernel. Non-symmetric relationships (e.g. AV, 1C) should not be decomposed using this method, nor should any assumptions be made about different kernels (e.g. the 1C relationship would appropriately be given the value 2, but not 4)
`output`	character. Denotes what kind of output to return. If 'confs', a vector of 95% confidence intervals. if 'vect', a vector of all permuted axial value results

Value

If ouput = 'confs', returns a `numeric vector` of 95% confidence intervals and mean axial value. if output = 'vect', returns a `numeric vector` of all permuted axial value results

See Also

Other axial_helpers: `axials()`, `axials_add()`, `axials_decompose()`, `axials_subtract()`, `axpermute_subtract()`

Examples

```
po_dists <- rexp(100, 1 / 50)
axpermute(po_dists, composite = 1)
```

`axpermute_standard` *Calculate the intergenerational (PO) dispersal kernel from the distributions of close kin (bootstrapped)*

Description

This function takes (at least) two vectors of kinship dispersal distances from defined kinship categories, and returns a resulting calculation of the parent-offspring (intergenerational) kinship dispersal kernel with bootstrapped confidence intervals. Dispersal distances can be inputted as numeric vectors, or alternatively as objects of classes `KinPairData` or `KinPairSimulation`.

Usage

```
axpermute_standard(
  avect = NULL,
  bvect = NULL,
  acat = NULL,
  bcat = NULL,
  nreps = 1000,
  nsamp = "std",
  amix = FALSE,
  bmix = FALSE,
  amixcat = NULL,
  bmixcat = NULL,
  acomp = FALSE,
  bcomp = FALSE,
  acompvect = NULL,
  bcompvect = NULL,
  acompcat = NULL,
  bcompcat = NULL,
  acycle = NULL,
  bcycle = NULL,
  amixcycle = NULL,
  bmixcycle = NULL,
  acompcycle = NULL,
  bcompcycle = NULL,
  output = "confs",
  override = FALSE
)
```

Arguments

`avect`	vector a of kin dispersal distances for the less closely related kinship category OR object of class KinPairData.
`bvect`	vector b of kin dispersal distances for the more closely related kinship category OR object of class KinPairData.

acat	kinship category of kin dispersal vector avect. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
bcat	kinship category of kin dispersal vector bvect. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
nreps	number of permutations to run for confidence intervals (default 1000)
nsamp	number of kin pairs to subsample for each permutation. Either "std" or an integer. If "std" will be computed as equal to the sample size. (default "std")
amix	logical describing whether vector a is a mixture of two kinship categories. Used with amixcat. Default FALSE.
bmix	logical describing whether vector b is a mixture of two kinship categories. Used with bmixcat. Default FALSE.
amixcat	mixture kinship category of vector a. Must be set if amix == TRUE. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
bmixcat	mixture kinship category of vector b. Must be set if bmix == TRUE. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
acomp	logical denoting whether vector a should be composited with an additional kinship category vector. Used with acompvect and acompcat. Default FALSE.
bcomp	logical denoting whether vector b should be composited with an additional kinship category vector. Used with bcompvect and bcompcat. Default FALSE.
acompvect	vector acomp of kin dispersal distances for compositing with vector a OR object of class KinPairData. Must be set if acomp == TRUE.
bcompvect	vector bcomp of kin dispersal distances for compositing with vector b OR object of class KinPairData. Must be set if bcomp == TRUE.
acompcat	kinship category of kin dispersal vector acompvect. Must be set if acomp == TRUE. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
bcompcat	kinship category of kin dispersal vector bcompvect. Must be set if bcomp == TRUE. Must be one of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV", "HGAV", "H1C", "H1C1", "H2C"
acycle	breeding cycle number of kin dispersal vector avect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.
bcycle	breeding cycle number of kin dispersal vector bvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`amixcycle` breeding cycle number of kin dispersal vector amixvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`bmixcycle` breeding cycle number of kin dispersal vector bmixvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`acompcycle` breeding cycle number of kin dispersal vector acompvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`bcompcycle` breeding cycle number of kin dispersal vector bcompvect. Must be a nonnegative integer. (0, 1, 2, ...). Represents the number of complete breeding cycles the sampled individual has undergone before the checkpoint, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.

`output` string denoting what kind of output to return. If 'confs', a vector of 95% confidence intervals. if 'vect', a vector of all permutated axial value results

`override` whether or not to override the default -1 cycle compatibility check (default FALSE) override in situations where you are confident e.g. a c(-1, -1) cycle FS or HS category is truly zeroed (& thus separated from later stages by a complete lifespan)

Details

This (with its paired function `axials_standard`) are the core functions implemented in the `kindisperse` package. They enable the decomposition of the pedigree & dispersal information contained in the sampled distributions of close kin dyads (full siblings, first cousins, etc.) & its leveraging within an additive dispersal framework to estimate the key intergenerational (parent-offspring) dispersal parameter of a population. Four key ideas underpin the approach in this function: (a) tracing dispersal pedigrees to determine the number of complete intergenerational (breeding-cycle-spanning) dispersal events separating the sampled close-kin dyads; (b) using kin categories that share the same overarching kinship 'phase' to control for residual 'phased' (non-intergenerational) disperal events that occur at the pedigree branch point (e.g. ovipositional dispersal for full sibling mosquitoes), and (c) using synced or equivalent sampling points to eliminate non-intergenerational dispersal at the branch-tips of the pedigrees, then finally (d) decomposing the 'pure' pedigree-associated (intergenerational) dispersal into an estimae of the single-generation intergenerational dispersal parameter.

At its most basic, this function requires information about two dispersal vectors, a & b - both of a phased kinship category, & `vector a` having a more dispersed pedigree than `vector b`. In addition

to this initial pair of dispersed kin categories, either one or another matched pair of kin categories can be added:

1. A mixture category. This redefines the vector it is paired with (either a or b) so that rather than being considered as a 'pure' pedigree variant, it is considered as mixed with a different kin category, often of a differing pedigree phase. If used, the other initial vector must also be paired with a related mixture category or composite vector.
2. A composite dispersal vector. This is defined exactly as the initial dispersal vectors. After calculation, the axial value found is composited with that of the matched initial vector, and its kinship category redefined as a mixture category as above. If used, the other initial vector must also be paired with a related mixture category or composite vector. These can be paired so that a mixture category (e.g. first & half-first cousins where these could not be separated with available genetic data) can be counterbalanced with the composition of full sibling & half-sibling dyads, which (assuming equal mixture) approximately controls for the phasing of the mixed kin categories, enabling an estimate of intergenerational dispersal without exact knowledge of the composition of the cousins distribution.

Each vector or `KinPairData` / `KinPairSimulation` object is paired with several other parameters: (1) a logical (e.g. `amix` delineating whether the category is being used in the calculation, (2) a category parameter (.e.g `acat`) defining what kin relationship is being measured, (3) an optional breeding cycle number (e.g. `acycle`) showing the number of breeding cycles each member of the kin pair has passed through before being sampled (the cycle vector `c(1, 0)` corresponds to an adult & a juvenile being sampled at the same point in the breeding cycle; `c(1, 1)` represents two adults (i.e. after their first breeding), etc.) . If a `KinPairData` or `KinPairSimulation` object is inputted, all paired parameters that are not explicitly set will default to those contained in the objects (using KinPair objects is the ideal way to deploy this function).

Confidence intervals are assigned via bootstrapping, or optionally the vector of all bootstrapped results can be outputted by setting `output` to `'vect'`, enabling its passing to other functions or external statistical analysis.

For further information on this function, package & the dispersal estimation method it represents, see the paper by Jasper et al. - "A genomic approach to inferring kinship reveals limited intergenerational dispersal in the yellow fever mosquito", doi:10.1111/17550998.13043.

Value

If output = 'confs' returns vector of 95% confidence intervals (with mean). If output = 'vect' returns vector of individual axial estimates from each permutation

See Also

Other axstandard: `axials_standard()`

Examples

```
cous <- rexp(100, 1 / 100)
fullsibs <- rexp(50, 1 / 50)
axpermute_standard(cous, fullsibs, acat = "1C", bcat = "FS")
```

`axpermute_subtract` *Subtract axial distributions with confidence intervals*

Description

Finds the difference between two different empirical axial distributions with confidence intervals. This is most useful when one distribution subsumes another and includes a unique dispersal event that needs to be extracted. For example, the FS category is subsumed by the 1C category, which can be written 'FS + PO'. In this circumstance, subtracting FS from 1C will yield an estimate of the PO kernel (the basic intergenerational dispersal kernel). Confidence intervals are assigned via bootstrapping, or optionally the vector of all bootstrapped results can be outputted by setting `output` to `'vect'`, enabling its passing to other functions or external statistical analysis.

Usage

```
axpermute_subtract(
  bigvals,
  smallvals,
  nreps = 1000,
  nsamp = "std",
  composite = 2,
  output = "confs"
)
```

Arguments

`bigvals`	numeric. Vector of distance distributions of the larger (subsuming) distribution (e.g. 1C) OR object of class KinPairData.
`smallvals`	numeric. Vector of distance distributions of the smaller (subsumed) distribution (e.g. FS) OR object of class KinPairData.
`nreps`	numeric. Number of permutations to perform when generating confidence intervals.
`nsamp`	numeric. number of kin pairs to subsample for each permutation. Either "std" or an integer. If "std" will be computed as equal to the sample size. (default "std")
`composite`	numeric. The number of separate 'draws' (dispersal events) from the kernel required to produce the final positions of the measured individuals. For example, the displacement of a child from parent at the same lifestage would involve 1 draw and thus be composite = 1. Two full siblings would be two draws (composite = 2) from the FS kernel. Non-symmetric relationships (e.g. AV, 1C) should not be decomposed using this method, nor should any assumptions be made about different kernels (e.g. the 1C relationship would appropriately be given the value 2, but not 4)
`output`	character. What kind of output to return. Either 'confs' (default -> confidence intervals) or 'vect -> vector of axial distances

Value

If output = 'confs' returns `numeric vector` of 95% confidence intervals and mean axial value. If output = 'vect' returns `numeric vector` of individual axial estimates from each permutation

See Also

Other axial_helpers: `axials()`, `axials_add()`, `axials_decompose()`, `axials_subtract()`, `axpermute()`

Examples

```
firstcous <- rexp(100, 1 / 80)
fullsibs <- rexp(100, 1 / 50)
axpermute_subtract(firstcous, fullsibs)
```

---

breeding_cycle *Access breeding cycle at sampling of* `DispersalModel` *object.*

Description

Access breeding cycle at sampling of `DispersalModel` object.

Usage

```
breeding_cycle(x)

## S4 method for signature 'DispersalModel'
breeding_cycle(x)

## S4 method for signature 'KinPairData'
breeding_cycle(x)
```

Arguments

`x`	object of class `DispersalModel`
`DispersalModel`	object of class `DispersalModel`
`KinPairData`	object of class `KinPairData`

Value

`integer(s) >= -1` Breeding cycle numbers of modeled dispersed kin. Represents the number of complete breeding cycles each indivdiual has undergone before the sampling point, where the time between birth and first reproduction is coded as `0`, that between first and second reproduction `1`, etc.

Methods (by class)

- `breeding_cycle(DispersalModel)`:
- `breeding_cycle(KinPairData)`:

breeding_stage *Access life stage at which breeding occurs of* DispersalModel *object*

Description

Access life stage at which breeding occurs of DispersalModel object

Usage

```
breeding_stage(x)

## S4 method for signature 'DispersalModel'
breeding_stage(x)
```

Arguments

x	object of class DispersalModel
DispersalModel	object of class DispersalModel

Value

character life stage at which breeding occurs for modeled dispersed kin.

Methods (by class)

- breeding_stage(DispersalModel):

check_valid_kinship *Check valid kinship*

Description

Checks if vector of kinship categories contains all valid entries

Usage

```
check_valid_kinship(vect)
```

Arguments

vect	vector of kinship categories

Value

TRUE if valid. Error otherwise.

check_valid_lifestage *Check valid lifestage*

Description

Checks if vector of lifestages contains all valid entries

Usage

```
check_valid_lifestage(vect)
```

Arguments

vect	vector of lifestages

Value

TRUE if valid. Error otherwise

csv_to_kinpair *Reads* .csv *and converts to* KinPairData *object*

Description

This function is part of suite of functions handling file import/export for kinship dispersal objects.

.csv & .tsv reading functions at minimum require the .delim file to contain a column titled 'distance' containing distances between kin pairs. It can optionally contain a column of kinship values 'kinship' as well as a column of lifestage values 'lifestage'. If the file contains more than one value in the kinship or lifestage columns (e.g. bot 'FS' and 'HS') - the corresponding function parameter must be set to pick a corresponding subset of dispersed pairs. where parameters are set in the absence of file columns, these values are assigned to the returned KinPairData object.

Usage

```
csv_to_kinpair(file, kinship = NULL, lifestage = NULL, ...)
```

Arguments

file	The file path to read from
kinship	character. kin category to assign or extract from data. one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C , H1C1 or H2C
lifestage	character. lifestage to assign or extract from data. one of 'unknown', 'immature' or 'ovipositional'.
...	additional arguments to pass to read_csv

Value

returns an object of class KinPairData

See Also

Other import_functions: df_to_kinpair(), read_kindata(), tsv_to_kinpair(), vector_to_kinpair()

---

df_to_kinpair *Convert dataframe or tibble to KinPairData class*

Description

This function at minimum requires the dataframe to contain a column titled 'distance' containing distances between kin pairs. It can optionally contain a column of kinship values 'kinship' as well as a column of lifestage values 'lifestage'. If the file contains more than one value in the kinship or lifestage columns (e.g. bot 'FS' and 'HS') - the corresponding function parameter must be set to pick a corresponding subset of dispersed pairs. where parameters are set in the absence of file columns, these values are assigned to the returned KinPairData object.

Usage

```
df_to_kinpair(data, kinship = NULL, lifestage = NULL, lifecheck = TRUE)
```

Arguments

data	data.frame or tibble of kin distances - can contain $distance (kin distances), $kinship (kin cats) & $lifestage columns
kinship	character. kin category to assign or extract from data. one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C , H1C1 or H2C
lifestage	character. lifestage to assign or extract from data. one of 'unknown', 'immature' or 'ovipositional'.
lifecheck	logical. If TRUE (default) tests if lifestage is valid, if FALSE, ignores this test. Set to FALSE when using custom lifestages.

Value

returns valid KinPairData object

See Also

Other import_functions: csv_to_kinpair(), read_kindata(), tsv_to_kinpair(), vector_to_kinpair()

Examples

```
mydata <- tibble::tibble(
  distance = 1:10, lifestage = "immature",
  kinship = c("FS", "FS", "FS", "FS", "FS", "FS", "HS", "HS", "HS", "HS")
)
df_to_kinpair(mydata, kinship = "FS")
```

DispersalModel-class *DispersalModel Class*

Description

The class `DispersalModel` is an S4 Class supplying organism-specific information about dispersal stages (with axial sigmas), FS & HS branch points, and the dispersal stage at which sampling occurs.It is used with the `simulate_kindist_custom` function to enable the simulation of uniquely defined breeding & dispersal cycles.

Usage

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

## S4 method for signature 'DispersalModel'
initialize(
  .Object,
  stages = NULL,
  dispersal_vector = NULL,
  fs = NULL,
  hs = NULL,
  sampling_stage = NULL,
  cycle = NULL,
  breeding_stage = NULL,
  visible_stage = NULL
)
```

Arguments

`object`	an object of class `DispersalModel`
`.Object`	object to be constructed into DispersalModel class
`stages`	character. Ordered vector of all dispersal stages across the breeding cycle of the modeled species
`dispersal_vector`	numeric. Named vector of custom breeding cycle stages and their corresponding axial dispersal values
`fs`	character. breeding cycle stage at which first substantial FS-phased dispersal occurs

`hs` character. breeding cycle stage at which first substantial HS-phased dispersal occurs

`sampling_stage` character. stage in the breeding cycle at which samples are to be collected for kin identification.

`cycle` non-negative integer. Breeding cycle numbers of dispersed kin to be modeled. Represents the number of complete breeding cycles each simulated individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0)

`breeding_stage` (character) - stage in the cycle at which breeding occurs. Must correspond to a previously described cycle stage name. By default, equated with the .HS stage. This stage corresponds to the **generation** of next-generation individuals; the .FS & .HS stages correspond to their separation. Needed for situations where individuals are sampled before they separate from the parent. Modify if the modeled .HS gamete dispersal event does not correspond to the initial breeding event.

`visible_stage` (character) - stage in the cycle at the **beginning** of which individuals are visible to the study for sampling rather than their parents (i.e. the beginning point of `cycle 0`). By default, equated with the `fs` stage. This parameter determines how many dispersal stages individuals have gone through before they are sampled - if `.sampling_stage` occurs just **after** `.visible_stage`, the sampled individuals will have dispersed through only a small amount of the breeding cycle. if `.sampling_stage` occurs just **before** `.visible_stage`, the sampled individuals will have dispersed throughout most of the breeding cycle before being sampled. If `.cycle` is set to `-1`, dispersal stages between breeding & visibility can be accessed.

`DispersalModel` an object of class DispersalModel

Details

The original simulation functions in this package (`simulate_kindist_simple()` & `simulate_kindist_composite`) were designed for an organism with a specific (& relatively simple) breeding & dispersal cycle. 'simple' corresponded to a single dispersal event across a lifespan, equivalency of all dispersal phases (FS, HS, PO) and no lifetime overlaps. 'composite' corresponded to many insect dispersal situations, where breeding & oviposition are the key 'phase-defining' events (i.e., they lead to the initial gamete dispersal of half siblings & full siblings from each other), where field sampling typically occurs via ovitraps

More general dispersal scenarios (e.g in mammals) require the ability to uniquely specify a variety of distinct breeding ecologies & sampling schemes: the `DispersalModel` class paired with the `simulate_kindist_custom` function achieves this by defining a breeding cycle with an arbitrary number of dispersal phases (the `dispersal_vector` slot, accessed by the `dispersal_vector` method).

The breeding structure of a species may also impact at which stage FS and HS phase branches occur. In *Ae. aegypti*, males mate with multiple females in a (single) breeding season, and a female typically carried the egg of only one male. In this context the FS (full-sibling) phase would be set to correspond to the female's oviposition dispersal, while the HS (half-sibling) phase would be set to correspond to the male's breeding dispersal (as its gametes will then be dispersed by multiple

females across their gravid & ovipositional phases). However, in e.g. some species of the marsupial *Antechinus*, the FS branch point would be more appropriately associated with juveniles at the time that they leave the mother's pouch. The `fs` and `hs` slots & accessor functions enable the assignment of these phase branches to any defined life phase. Similarly, the `sampling_stage` slot & method allow the sampling point to be set to correspond to any phase of the defined breeding cycle.

The next parameter stored in this object is the breeding cycle number `cycle`, accessed by the `breeding_cycle` method. This parameter enables the treatment of species that undergo multiple breeding cycles in one lifetime. This is defined as a length two vector describing the number of breeding cycles undergone by the final descendant of branch 1 and branch 2 of the dispersal pedigree before their sampling (or after branching in the case of PO). (where branch one is the 'senior' and branch two the 'junior' member of the pedigree) (so uncle is branch one, nephew branch two, grandmother branch one, granddaughter branch two, etc.). For each member of the resulting kin pair, the cycle number represents the number of complete breeding cycles each individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. This enables an application of the simulation functions defined here to deal with populations with some amount of overlap between generations.

Note that this 'breeding cycle' approach is only applicable in situations where there is an approximate equivalence between the dispersal which occurs in the first 'juvenile' breeding cycle and that which occurs between later breeding cycles. This parameter is implemented here, but it will often be more productive to implement it instead as a parameter of the `simulate_kindist_custom` function (the cycle parameter there if set overrides whatever was defined within this object)

The final parameter stored in this object is the breeding stage, `breeding_stage`. This describes the stage at which the descendant individuals are generated (as opposed to `fs` & `hs`, which describe the point at which they are dispersed from the parent)

Value

returns object of class `DispersalModel`

No return value. Called for side effects

returns an object of class `DispersalModel`

Methods (by generic)

- `show(DispersalModel)`: print method
- `initialize(DispersalModel)`: initialization method

Slots

`dispersal_vector` numeric. Named vector of custom breeding cycle stages and their corresponding axial dispersal values

`stages` character. Ordered vector of all dispersal stages across the breeding cycle of the modeled species

`fs` character. breeding cycle stage at which first substantial FS-phased dispersal occurs

`hs` character. breeding cycle stage at which first substantial HS-phased dispersal occurs

`sampling_stage` character. stage in the breeding cycle at which samples are to be collected for kin identification.

`cycle` non-negative integer. Breeding cycle numbers of dispersed kin to be modeled. Represents the number of complete breeding cycles each individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0)

`breeding_stage` (character) - stage in the cycle at which breeding occurs. Must correspond to a previously described cycle stage name. By default, equated with the .HS stage. This stage corresponds to the **generation** of next-generation individuals; the .FS & .HS stages correspond to their separation. Needed for situations where individuals are sampled before they separate from the parent. Modify if the modeled .HS gamete dispersal event does not correspond to the initial breeding event.

`visible_stage` (character) - stage in the cycle at the **beginning** of which individuals are visible to the study for sampling rather than their parents (i.e. the beginning point of `cycle 0`). By default, equated with the `fs` stage. This parameter determines how many dispersal stages individuals have gone through before they are sampled - if `.sampling_stage` occurs just **after** `.visible_stage`, the sampled individuals will have dispersed through only a small amount of the breeding cycle. if `.sampling_stage` occurs just **before** `.visible_stage`, the sampled individuals will have dispersed throughout most of the breeding cycle before being sampled. If `.cycle` is set to `-1`, dispersal stages between breeding & visibility can be accessed.

See Also

Other kdclasses: `KinPairData-class`, `KinPairSimulation-class`

`dispersal_model` *Create Dispersal Model of an Organism*

Description

The function creates an object of class `DispersalModel` carrying organism-specific information about dispersal stages (with axial sigmas), FS & HS branch points, and the dispersal stage at which sampling occurs.It is used with the `simulate_kindist_custom` function to enable the simulation of uniquely defined breeding & dispersal cycles.

Usage

```
dispersal_model(
  ...,
  .FS = 0,
  .HS = .FS,
  .sampling_stage = 0,
  .cycle = 0,
  .breeding_stage = .HS,
  .visible_stage = .FS
)
```

Arguments

`...`	name, value (numeric) pairs pairing custom lifestages with their corresponding axial dispersal values. MUST be in chronological order across the entire breeding cycle.
`.FS`	(character) - breeding cycle stage at which first substantial FS-phased dispersal occurs. Must correspond to a previously described cycle stage name. Typically reflects the first dispersal of female gametes from the mother at (variously) egg-laying, birth, weaning stages (species-dependent). Use care in adapting to situations where multiple breeding and/or dispersal routes commonly lead to the FS phase
`.HS`	(character) - breeding cycle stage at which first substantial HS-phased dispersal occurs. Must correspond to a previously described cycle stage name. Typically reflects the movement of male gametes at e.g. the breeding stage (use care in adapting to situations where multiple dispersal routes commonly lead to the HS phase)
`.sampling_stage`	(character) - stage in the breeding cycle at which samples are to be collected for kin identification. Must correspond to a previously described cycle stage name. (so collection of eggs corresponds to an egg-laying stage, as juveniles to a juvenile stage, etc.)
`.cycle`	(integer >= -1 or vector of two such integers) breeding cycle numbers of dispersed kin to be modeled. Represents the number of complete breeding cycles each simulated individual has undergone before the sampling point, where the time between first dispersal and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). If .cycle is specially set to '-1' this constitutes the sampling of an individual before it has differentiated (via dispersal) from the parent. Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan. As the rest of the model is compatible with a variety of cycle points, this parameter will often be overridden by the 'cycle' parameter in the `simulate_kindist_custom` function.
`.breeding_stage`	(character) - stage in the cycle at which breeding occurs. Must correspond to a previously described cycle stage name. By default, equated with the .HS stage. This stage corresponds to the **generation** of next-generation individuals; the .FS & .HS stages correspond to their separation. Needed for situations where individuals are sampled before they separate from the parent. Modify if the modeled .HS gamete dispersal event does not correspond to the initial breeding event.
`.visible_stage`	(character) - stage in the cycle at the **beginning** of which individuals are visible to the study for sampling rather than their parents (i.e. the beginning point of `cycle 0`). By default, equated with the .FS stage. This parameter determines how many dispersal stages individuals have gone through before they are sampled - if `.sampling_stage` occurs just **after** `.visible_stage`, the sampled individuals will have dispersed through only a small amount of the breeding cycle. if `.sampling_stage` occurs just **before** `.visible_stage`, the sampled individuals will have dispersed throughout most of the breeding cycle before being

sampled. If `.cycle` is set to `-1`, dispersal stages between breeding & visibility can be accessed.

Details

The original simulation functions in this package (`simulate_kindist_simple()` & `simulate_kindist_composite`) were designed for an organism with a specific (& relatively simple) breeding & dispersal cycle. 'simple' corresponded to a single dispersal event across a lifespan, equivalency of all dispersal phases (FS, HS, PO) and no lifetime overlaps. 'composite' corresponded to many insect dispersal situations, where breeding & oviposition are the key 'phase-defining' events (i.e., they lead to the initial gamete dispersal of half siblings & full siblings from each other), where field sampling typically occurs via ovitraps

More general dispersal scenarios (e.g in mammals) require the ability to uniquely specify a variety of distinct breeding ecologies & sampling schemes: the `DispersalModel` class paired with the `simulate_kindist_custom` function achieves this by defining a breeding cycle with an arbitrary number of dispersal phases (the `dispersal_vector` slot, accessed by the `dispersal_vector` method).

The breeding structure of a species may also impact at which stage FS and HS phase branches occur. In *Ae. aegypti*, males mate with multiple females in a (single) breeding season, and a female typically carried the egg of only one male. In this context the FS (full-sibling) phase would be set to correspond to the female's oviposition dispersal, while the HS (half-sibling) phase would be set to correspond to the male's breeding dispersal (as its gametes will then be dispersed by multiple females across their gravid & ovipositional phases). However, in e.g. some species of the marsupial *Antechinus*, the FS branch point would be more appropriately associated with juveniles at the time that they leave the mother's pouch. The `.FS` and `.HS` parameters enable the assignment of these phase branches to any defined life phase. Similarly, the `.sampling_stage` parameter allow the sampling point to be set to correspond to any phase of the defined breeding cycle (this is later accessed with the `sampling_stage` method).

The final parameter stored in this object is the breeding cycle number `.cycle`, accessed later by the `breeding_cycle` method. This parameter enables the treatment of species that undergo multiple breeding cycles in one lifetime. This is defined as a length two vector describing the number of breeding cycles undergone by the final descendant of branch 1 and branch 2 of the dispersal pedigree before their sampling. (where branch one is the 'senior' and branch two the 'junior' member of the pedigree) (so uncle is branch one, nephew branch two, grandmother branch one, granddaughter branch two, etc.). For each member of the resulting kin pair, the cycle number represents the number of complete breeding cycles each individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. This enables an application of the simulation functions defined here to deal with populations with some amount of overlap between generations.

Note that this 'breeding cycle' approach is only applicable in situations where there is an approximate equivalence between the dispersal which occurs in the first 'juvenile' breeding cycle and that which occurs between later breeding cycles. This parameter is implemented here, but it will often be more productive to implement it instead as a parameter of the `simulate_kindist_custom` function (the cycle parameter there if set overrides whatever was defined within this object)

Value

Returns an object of class DispersalModel containing custom lifestages and dispersal, phase & sampling parameters that can be passed to simulation functions.

Examples

```
antechinus_model <- dispersal_model(pouch = 25, nest = 25, free_living = 250, breeding = 40,
gestation = 25, .FS = "nest", .HS = "breeding", .sampling_stage = "nest")
antechinus_model
```

---

dispersal_vector *Access dispersal vector of* DispersalModel *object.*

Description

Access dispersal vector of DispersalModel object.

Usage

```
dispersal_vector(x)

## S4 method for signature 'DispersalModel'
dispersal_vector(x)
```

Arguments

x	object of class DispersalModel
DispersalModel	object of class DispersalModel

Value

numeric vector named vector of custom lifestages & associated dispersal sigmas.

Methods (by class)

- dispersal_vector(DispersalModel):

display_appdata *Show printout of named items stored in appdata.*

Description

This function is part of a suite of functions handling the interface between the kindisperse app & R package. Due to how shiny's interactive programming works, ordinary objects are not visible to the reactive functions embedded in the app. The solution implemented here is to construct a custom environment, `env_appdata`, that is accessible within the app and outside of it.

This function prints a summary of all objects currently stored within the app interface environment, by name and class

Usage

```
display_appdata()
```

Value

No return value, called for side effects

See Also

Other app_ports: `mount_appdata()`, `reset_appdata()`, `reset_tempdata()`, `retrieve_appdata()`, `retrieve_tempdata()`, `retrieveall_appdata()`, `unmount_appdata()`

Examples

```
mount_appdata(kin_pair_data(), "my_kindata")
mount_appdata(simulate_kindist_simple(nsims = 10), "my_simdata")

display_appdata()
```

distances *Access or assign* `distances` *category of* `KinPairData` *class objects*

Description

Access or assign `distances` category of `KinPairData` class objects

Usage

```
distances(x)

## S4 method for signature 'KinPairData'
distances(x)
```

Arguments

x	object of class KinPairData
KinPairData	object of class KinPairData

Value

Returns a numeric vector of kin separation distances

Methods (by class)

- distances(KinPairData):

See Also

Other kpdmethods: kinship(), lifestage()

---

elongate *Change the shape (aspect ratio) of a rectangle while preserving area*

Description

This function is used to manipulate the dimensions parameter in other package functions, which control site dimentions. These geometries can be entered innto functions in a few ways: (a) a single numeric value, which will be interpreted as the length of the side of a square; (b) a numeric vector of length two, which will be interpreted as the length & width of the sample site; (c) either of the above passed to this function, which takes the rectangular site dimensions and alters their aspect ratio (ratio of length to width) while preserving the underlying area the study site covers.

Usage

```
elongate(dims, aspect = 1)
```

Arguments

dims	Original rectangle dimensions - either single number (length of side of square) or length 2 numeric vector (lengths of sides x and y of rectangle)
aspect	Aspect ratio of side lengths x & y (i.e. x/y) in the new rectangle

Value

Returns a numeric vector containing the side lengths c(x, y) of a transformed rectangle with preserved area

Examples

```
elongate(10, 100)
elongate(c(5, 125), 4)
```

filtertype *Access filtertype of* KinPairSimulation *object*

Description

Access filtertype of KinPairSimulation object

Usage

```
filtertype(x)

filtertype(x) <- value

## S4 method for signature 'KinPairSimulation'
filtertype(x)
```

Arguments

x	object of class KinPairSimulation
value	new value to assign
KinPairSimulation	object of class KinPairSimulation

Value

character filter status of simulation

returns a modified object of the relevant class

character filter status of KinPairSimulation object

Methods (by class)

- filtertype(KinPairSimulation):

filter_methods *Access or modify the* filter *parameters of* KinPairSimulation *objects*

Description

These generics & methods work as an interface between KinPairSimulation objects and the sample_kindist function. They either retrieve the value of pre-existing filter steps that have been applied to the object (e.g. upper(x)) or assign such a filtering parameter to the KinPairSimulation object (e.g. sampledims(x) <- value). In this case, the method passes the KinPairSimulation object to the sample_kindist() function for subsampling or filtering, then updates the sampling parameter before returning the modified object. Note that while the sample_kindist function can take KinPairData objects, the methods described here are only applicable to objects of class KinPairSimulation.

Usage

```
upper(x)

upper(x) <- value

lower(x)

lower(x) <- value

spacing(x)

spacing(x) <- value

samplenum(x)

samplenum(x) <- value

sampledims(x)

sampledims(x) <- value

## S4 method for signature 'KinPairSimulation'
upper(x)

## S4 method for signature 'KinPairSimulation'
lower(x)

## S4 method for signature 'KinPairSimulation'
spacing(x)

## S4 method for signature 'KinPairSimulation'
samplenum(x)

## S4 method for signature 'KinPairSimulation'
sampledims(x)

## S4 replacement method for signature 'KinPairSimulation'
upper(x) <- value

## S4 replacement method for signature 'KinPairSimulation'
lower(x) <- value

## S4 replacement method for signature 'KinPairSimulation'
spacing(x) <- value

## S4 replacement method for signature 'KinPairSimulation'
samplenum(x) <- value
```

```
## S4 replacement method for signature 'KinPairSimulation'
sampledims(x) <- value
```

Arguments

x	object of class KinPairSimulation
value	value for parameter to be adjusted to
KinPairSimulation	object of class KinPairSimulation

Value

either the accessed numeric filter parameter or a filtered KinPairSimulation object

Functions

- upper(KinPairSimulation):
- lower(KinPairSimulation):
- spacing(KinPairSimulation):
- samplenum(KinPairSimulation):
- sampledims(KinPairSimulation):
- upper(KinPairSimulation) <- value:
- lower(KinPairSimulation) <- value:
- spacing(KinPairSimulation) <- value:
- samplenum(KinPairSimulation) <- value:
- sampledims(KinPairSimulation) <- value:

See Also

Other kpsmethods: access_sigmas, kernelshape(), kerneltype(), simtype()

---

fs *Access FS phase split point of* DispersalModel *object.*

Description

Access FS phase split point of DispersalModel object.

Usage

```
fs(x)

## S4 method for signature 'DispersalModel'
fs(x)
```

Arguments

x	object of class DispersalModel
DispersalModel	object of class DispersalModel

Value

character FS phase split

Methods (by class)

- fs(DispersalModel):

---

get_dispersal_model *Access dispersal model of* KinPairSimulation *object*

Description

Access dispersal model of KinPairSimulation object

Usage

```
get_dispersal_model(x)

## S4 method for signature 'KinPairSimulation'
get_dispersal_model(x)
```

Arguments

x	object of class KinPairSimulation
KinPairSimulation	object of class KinPairSimulation

Value

returns an object of class DispersalModel

Methods (by class)

- get_dispersal_model(KinPairSimulation):

hs *Access HS phase split point of* DispersalModel *object.*

Description

Access HS phase split point of DispersalModel object.

Usage

```
hs(x)

## S4 method for signature 'DispersalModel'
hs(x)
```

Arguments

x	object of class DispersalModel
DispersalModel	object of class DispersalModel

Value

character HS phase split

Methods (by class)

- hs(DispersalModel):

is.DispersalModel *Check if object is of class* DispersalModel

Description

Check if object is of class DispersalModel

Usage

```
is.DispersalModel(x)
```

Arguments

x	object to be checked

Value

returns TRUE if of class DispersalModel, FALSE if not

is.KinPairData *Check if object is of class KinPairData*

Description

Check if object is of class KinPairData

Usage

```
is.KinPairData(x)
```

Arguments

x	object to be checked

Value

Returns TRUE if of class `KinPairData`, FALSE if not.

is.KinPairSimulation *Check if object is of class KinPairSimulation*

Description

Check if object is of class KinPairSimulation

Usage

```
is.KinPairSimulation(x)
```

Arguments

x	object to be checked

Value

Returns TRUE if of class KinPairSimulation, FALSE if not

kernelshape *Access kernel type of* KinPairSimulation *object*

Description

Access kernel type of KinPairSimulation object

Usage

```
kernelshape(x)

## S4 method for signature 'KinPairSimulation'
kernelshape(x)
```

Arguments

x	object of class KinPairSimulation
KinPairSimulation	object of class KinPairSimulation

Value

character the shape parameter used in kernel simulation (if kerneltype is vgamma)

character the shape parameter used in kernel simulation (if kerneltype is vgamma)

Methods (by class)

- kernelshape(KinPairSimulation):

See Also

Other kpsmethods: access_sigmas, filter_methods, kerneltype(), simtype()

kerneltype *Access or assign kerneltype of* KinPairSimulation *object*

Description

Access or assign kerneltype of KinPairSimulation object

Usage

```
kerneltype(x)

kerneltype(x) <- value

## S4 method for signature 'KinPairSimulation'
kerneltype(x)
```

Arguments

x	object of class KinPairSimulation
value	new value to assign
KinPairSimulation	object of class KinPairSimulation

Value

character the type of statistical kernel used to run the simulation (Gaussian, Laplace, vgamma)

returns a modified object of the relevant class with altered kerneltype parameter

character the type of statistical kernel used to run the simulation (Gaussian, Laplace, vgamma)

Methods (by class)

- kerneltype(KinPairSimulation):

See Also

Other kpsmethods: access_sigmas, filter_methods, kernelshape(), simtype()

KinPairData-class *Formal class KinPairData*

Description

The class KinPairData is a formal (S4) class for storing kinship and lifespan dispersal information concerning kin pairs. It is the base class on which the KinPairSimulation class is built. The KinPairData class is used to store information about the spatial distribution of kin dyads for use in calculating axial sigmas of intergenerational dispersal as initially implemented in Jasper et al. 2019 (doi:10.1111/17550998.13043).

Usage

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

## S4 method for signature 'KinPairData'
initialize(
  .Object,
  data = NULL,
  kinship = NULL,
  lifestage = NULL,
  cycle = NULL,
  ...
)
```

Arguments

object	an object of class KinpairData
.Object	the KinPairData object to be constructed
data	data about kinship to be used to construct object (tibble, data.frame, or numeric vector of distances)
kinship	character. Kinship category value for object. - one of PO, FS, HS, AV, HAV, GG, 1C, H1C, GAV, HGAV, 1C1, H1C1, GGG, 2C, and H2C.
lifestage	character. Lifestage value for object. - one of 'immature', 'ovipositional' or 'unknown'
cycle	non-negative integer or vector of two such integers - Represents the number of complete breeding cycles each simulated individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). If the first individual was sampled as a juvenile & the second as an adult of equivalent stage, the vector c(0, 1) would be used. In most situations, defualt will be appropriate
...	additional argument to pass to downstream functions in future
KinPairData	object of class KinPairData

Details

This class is essentially wrapped around the `tbl_df` class but with (a) expectations around certain columns that must be present (`id1`, `id2`, `kinship`, & `distance` - three 'character' & one 'numeric' column), as well as (b) additional attributes (`kinship`, `lifestage`, & `cycle`) characterizing the close-kin dyads being stored.These attributes, as well as the embedded vector of distances, can be accessed with the methods `kinship`, `lifestage`, `breeding_cycle` and `distances`.

Objects from this class are returned from the `df_to_kinpair` and `csv_to_kinpair` functions (& related), and are directly constructed with the namesake `KinPairData()` function. They can be passed to the `sample_kindist` function for filtering and subsampling, and to axial functions (including `axials_standard` and `axpermute_standard`) for estimation of axial dispersal.

Value

returns object of class `KinPairData`

No return value, called for side effects

Returns an object of class `KinPairData`

Methods (by generic)

- `show(KinPairData)`: standard print method
- `initialize(KinPairData)`: initialize method

Slots

`kinship` character - one of PO, FS, HS, AV, HAV, GG, 1C, H1C, GAV, HGAV, 1C1, H1C1, GGG, 2C, and H2C.

`lifestage` character - lifestage at sampling - either 'immature', 'ovipositional' or a stage corresponding to a `DispersalModel` custom stage

`cycle` non-negative integer or vector of two such integers - Represents the number of complete breeding cycles each individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). If the first individual was sampled as a juvenile & the second as an adult of equivalent stage, the vector c(0, 1) would be used. In most situations, the default will be appropriate

`tab` tbl_df. - tibble of dispersal values

See Also

Other kdclasses: `DispersalModel-class`, `KinPairSimulation-class`

`KinPairSimulation-class`

KinPairSimulation Class

Description

The class `KinPairSimulation` is a formal (S4) class for storing kinship and dispersal distribution information derived from simulations in the `kindisperse` package. It is derived from the `KinPairData` class. The `KinPairSimulation` class is used to store information about the spatial distribution of kin dyads for use in calculating axial sigmas of intergenerational dispersal as initially implemented in Jasper et al. 2019 (doi:10.1111/17550998.13043).

Usage

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

## S4 method for signature 'KinPairSimulation'
initialize(
  .Object,
  data = NULL,
  kinship = NULL,
  lifestage = NULL,
  simtype = NULL,
  kerneltype = NULL,
  kernelshape = NULL,
  posigma = NULL,
  initsigma = NULL,
  breedsigma = NULL,
  gravsigma = NULL,
  ovisigma = NULL,
  customsigma = NULL,
  cycle = NULL,
```

```
  simdims = NULL,
  call = NULL,
  filtertype = NULL,
  upper = NULL,
  lower = NULL,
  spacing = NULL,
  samplenum = NULL,
  sampledims = NULL,
  model = NULL
)
```

Arguments

object	object of class KinPairSimulation
.Object	object to be constructed into KinPairSimulation class
data	tbl_df. tibble of simulation values
kinship	character - one of PO, FS, HS, AV, HAV, GG, 1C, H1C, GAV, HGAV, 1C1, H1C1, GGG, 2C, and H2C.
lifestage	character - one of 'unknown', 'immature' or 'ovipositional'
simtype	character - simulation type
kerneltype	character. - 'Gaussian', 'Laplace' or 'vgamma' (variance-gamma)
kernelshape	numeric. - value of kernel shape of simulation (if using kernel with shape parameter e.g. vgamma)
posigma	numeric - overall value of dispersal sigma (for simple kernel)
initsigma	numeric. - value of pre-breeding dispersal sigma (for composite kernel)
breedsigma	numeric. - value of breeding dispersal sigma (for composite kernel)
gravsigma	numeric. - value of post-breeding dispersal sigma (for composite kernel)
ovisigma	numeric. - value of oviposition dispersal sigma (for composite kernel)
customsigma	numeric. - vector of named custom dispersal sigmas (for custom kernel)
cycle	integer - number of breeding cycles sampled individual has survived (for custom kernel)
simdims	numeric. - dimensions of sampling area (assumes one side of square)
call	call. Call to create object
filtertype	character. whether the initial sim has been further filtered
upper	numeric. - FILTER: upper threshold used
lower	numeric. - FILTER: lower threshold used
spacing	numeric. - FILTER: spacing used
samplenum	numeric. - FILTER: sample number used
sampledims	numeric. - FILTER: sample dimensions used
model	list - model information if custom simulation used to generate object
KinPairSimulation	an object of class KinPairSimulation

Details

This class is essentially wrapped around the `tbl_df` class but with (a) expectations around certain columns that must be present (`id1`, `id2`, `kinship`, & `distance` - three 'character' & one 'numeric' column), as well as (b) additional attributes (`kinship`, `lifestage`, & `cycle`) characterizing the close-kin dyads being stored.These attributes, as well as the embedded vector of distances, can be accessed with the methods `kinship`, `lifestage`, `breeding_cycle` and `distances`. In addition to the above attributes (derived from the `KinPairData` class), this class contains attributes capturing the simulation type & parameters used to generate the final distribution of kin dyads.

Objects from this class are returned from the `simulate_kindist_composite`, `simulate_kindist_simple` and `simulate_kindist_custom` functions (& related), and are directly constructed with the namesake `KinPairSimulation()` function. They can be passed to the `sample_kindist` function for filtering and subsampling, and to axial functions (including `axials_standard` and `axpermute_standard`) for estimation of axial dispersal.

Value

returns object of class `KinPairSimulation`

No return value, called for side effects

Returns an object of class `KinPairSimulation`

Methods (by generic)

- `show(KinPairSimulation)`: print method
- `initialize(KinPairSimulation)`: initialisation method

Slots

`kinship` character - one of PO, FS, HS, AV, HAV, GG, 1C, H1C, GAV, HGAV, 1C1, H1C1, GGG, 2C, and H2C.

`simtype` character. - one of 'simple', 'composite' or 'custom'

`kerneltype` character. - 'Gaussian', 'Laplace' or 'vgamma' (variance-gamma)

`posigma` numeric. - overall value of dispersal sigma (for simple kernel)

`initsigma` numeric. - value of pre-breeding dispersal sigma (for composite kernel)

`breedsigma` numeric. - value of breeding dispersal sigma (for composite kernel)

`gravsigma` numeric. - value of post-breeding dispersal sigma (for composite kernel)

`ovisigma` numeric. - value of oviposition dispersal sigma (for composite kernel)

`customsigma` numeric - vector of named custom dispersal sigmas (for custom kernel)

`simdims` numeric. - dimensions of sampling area (assumes 1 side of square)

`lifestage` character. - lifestage at sampling - either 'immature' or 'ovipositional'

`cycle` integer - number of breeding cycles sampled individuals have survived (for custom kernel)

`kernelshape` numeric. - shape parameter if vgamma kerneltype

`call` call. - call to create initial simulation

`tab` tbl_df. - tibble of simulation values

`filtertype` character. - whether the initial sim has been further filtered

`upper` numeric. - FILTER: upper threshold used

`lower` numeric. - FILTER: lower threshold used

`spacing` numeric. - FILTER: spacing used

`samplenum` numeric. - FILTER: sample number used

`sampledims` numeric. - FILTER: dimensions used

`model` `DispersalModel` - model of dispersal used to create object (with custom type)

See Also

Other kdclasses: `DispersalModel-class`, `KinPairData-class`

`KinPairSimulation_composite` *Constructor for KinPairSimulation Class (composite)*

Description

Constructor for KinPairSimulation Class (composite)

Usage

```
KinPairSimulation_composite(
  data = NULL,
  kinship = NULL,
  kerneltype = NULL,
  initsigma = NULL,
  breedsigma = NULL,
  gravsigma = NULL,
  ovisigma = NULL,
  simdims = NULL,
  lifestage = NULL,
  kernelshape = NULL,
  call = NULL,
  model = NULL
)
```

Arguments

`data`	tibble of pairwise kin classes & distances. Ideally contains fields id1 & id2 (chr) an distance (dbl) optionally includes coords (x1, y1, x2, y2), lifestage (ls1 & ls2), kinship (chr) and sims (dbl)
`kinship`	character. Code for kinship category of simulation. one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C or H2C

kerneltype	character. Statistical model for simulated dispersal kernel. Currently either "Gaussian", "Laplace" or "vgamma" (variance-gamma).
initsigma	numeric. Axial sigma of prebreeding ('juvenile') dispersal kernel (axial standard deviation).
breedsigma	numeric. Axial sigma of breeding dispersal kernel (axial standard deviation).
gravsigma	numeric. Axial sigma of post-breeding ('gravid') dispersal kernel (axial standard deviation).
ovisigma	numeric. Axial sigma of oviposition dispersal kernel (axial standard deviation).
simdims	numeric. Length of side of simulated area square.
lifestage	character. Simulated lifestage of sampling. Either "immature" (sampled at hatching) or "ovipositional" (sampled as an adult during oviposition - essentially one lifespan later than 'immature')
kernelshape	numeric. Value of shape parameter for simulated kernel if kernel requires one (e.g. vgamma kernel).
call	call object. Use to pass the system call that led to the generation of this class. (via sys.call)
model	DispersalModel - model information passed from simulation function

Value

Returns a KinPairSimulation Class object with simtype set to 'composite' and relevant fields included.

Examples

```
kindata <- tibble::tibble(
  id1 = c("a", "b", "c"), id2 = c("x", "y", "z"),
  distance = c(50, 45, 65), kinship = c("1C", "1C", "1C")
)
KinPairSimulation_composite(kindata,
  kinship = "1C", kerneltype = "Gaussian",
  initsigma = 15, breedsigma = 25, gravsigma = 20, ovisigma = 10, lifestage = "immature"
)
```

KinPairSimulation_custom

Constructor for KinPairSimulation Class (custom)

Description

Constructor for KinPairSimulation Class (custom)

Usage

```
KinPairSimulation_custom(
  data = NULL,
  kinship = NULL,
  kerneltype = NULL,
  customsigma = NULL,
  simdims = NULL,
  lifestage = NULL,
  kernelshape = NULL,
  cycle = NULL,
  call = NULL,
  model = NULL
)
```

Arguments

data	tibble of pairwise kin classes & distances. Ideally contains fields id1 & id2 (chr) an distance (dbl) optionally includes coords (x1, y1, x2, y2), lifestage (ls1 & ls2), kinship (chr) and sims (dbl)
kinship	character. Code for kinship category of simulation. one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C or H2C
kerneltype	character. Statistical model for simulated dispersal kernel. Currently either "Gaussian", "Laplace" or "vgamma" (variance-gamma).
customsigma	numeric. Named vector of custom breeding cycle stages and their corresponding axial dispersal values
simdims	numeric. Length of side of simulated area square.
lifestage	character. Simulated lifestage of sampling. Here, must correspond to a custom lifestage derived from 'customsigma'
kernelshape	numeric. Value of shape parameter for simulated kernel if kernel requires one (e.g. vgamma kernel).
cycle	non-negative integer. Breeding cycle numbers of dispersed kin to be modeled. Represents the number of complete breeding cycles each simulated individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0)
call	call object. Use to pass the system call that led to the generation of this class. (via sys.call)
model	DispersalModel - model information passed from simulation function

Value

Returns a KinPairSimulation Class object with simtype set to 'custom' and relevant fields included.

Examples

```
kindata <- tibble::tibble(
  id1 = c("a", "b", "c"), id2 = c("x", "y", "z"),
  distance = c(50, 45, 65), kinship = c("1C", "1C", "1C")
)
KinPairSimulation_custom(kindata,
  kinship = "1C", kerneltype = "Gaussian",
  customsigma = c(initsigma = 15, breedsigma = 25, gravsigma = 20, ovisigma = 10),
  lifestage = "ovisigma", cycle = 0
)
```

KinPairSimulation_simple

Constructor for KinPairSimulation Class (simple)

Description

Constructor for KinPairSimulation Class (simple)

Usage

```
KinPairSimulation_simple(
  data = NULL,
  kinship = NULL,
  kerneltype = NULL,
  posigma = NULL,
  simdims = NULL,
  lifestage = NULL,
  kernelshape = NULL,
  call = NULL,
  model = NULL
)
```

Arguments

data	tibble of pairwise kin classes & distances. Ideally contains fields id1 & id2 (chr) an distance (dbl) optionally includes coords (x1, y1, x2, y2), lifestage (ls1 & ls2), kinship (chr) and sims (dbl)
kinship	character. Code for kinship category of simulation. one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C or H2C
kerneltype	character. Statistical model for simulated dispersal kernel. Currently either "Gaussian", "Laplace" or "vgamma" (variance-gamma).
posigma	numeric. Axial sigma of dispersal kernel (axial standard deviation).
simdims	numeric. Length of side of simulated area square.

lifestage	character. Simulated lifestage of sampling. Either "immature" (sampled at hatching) or "ovipositional" (sampled as an adult during oviposition - essentially one lifespan later than 'immature')
kernelshape	numeric. Value of shape parameter for simulated kernel if kernel requires one (e.g. vgamma kernel).
call	call object. Use to pass the system call that led to the generation of this class. (via sys.call)
model	DispersalModel - model information passed from simulation function

Value

Returns a KinPairSimulation Class object with simtype set to 'simple' and relevant fields included.

Examples

```
kindata <- tibble::tibble(
  id1 = c("a", "b", "c"), id2 = c("x", "y", "z"),
  distance = c(50, 45, 65), kinship = c("1C", "1C", "1C")
)
KinPairSimulation_simple(kindata,
  kinship = "1C", kerneltype = "Gaussian",
  posigma = 38, lifestage = "immature"
)
```

---

kinpair_to_csv *Write* KinPairData *object to* .csv *format*

Description

This function is part of suite of functions handling file import/export for kinship dispersal objects. Writing to .csv or .tsv formats strips most KinPairData & KinPairSimulation class metadata and leaves a delimited file containing ids, kinship category, geographical distance, & x & y coordinates for each simulated pair. (removes class attributes)

Usage

```
kinpair_to_csv(x, file, ...)
```

Arguments

x	Object of class KinPairData or KinPairSimulation
file	The file path to write to
...	Additional arguments to pass to write_csv

Value

Invisibly returns the initial object

See Also

Other export_functions: `kinpair_to_tibble()`, `kinpair_to_tsv()`, `write_kindata()`

`kinpair_to_tibble` *Extract KinPairData class object to tibble*

Description

Extract `KinPairData` class object to tibble. Strips out most class metadata leaving a dataframe of disersal simulation data with a column added covering lifestage at sampling.

Usage

```
kinpair_to_tibble(x)
```

Arguments

`x`	object of class `KinPairData`

Value

tibble (class `tbl_df`)

See Also

Other export_functions: `kinpair_to_csv()`, `kinpair_to_tsv()`, `write_kindata()`

`kinpair_to_tsv` *Write* `KinPairData` *object to* `.tsv` *format*

Description

This function is part of suite of functions handling file import/export for kinship dispersal objects. Writing to .csv or .tsv formats strips most `KinPairData` & `KinPairSimulation` class metadata and leaves a delimited file containing ids, kinship category, geographical distance, & x & y coordinates for each simulated pair. (removes class attributes)

Usage

```
kinpair_to_tsv(x, file, ...)
```

Arguments

`x`	Object of class `KinPairData` or `KinPairSimulation`
`file`	The file path to write to
`...`	Additional arguments to pass to `write_tsv`

Value

Invisibly returns the initial object

See Also

Other export_functions: `kinpair_to_csv()`, `kinpair_to_tibble()`, `write_kindata()`

`kinship` *Access or assign* `kinship` *category of* `KinPairData` *class objects*

Description

Access or assign `kinship` category of `KinPairData` class objects

Usage

```
kinship(x)

kinship(x) <- value

## S4 method for signature 'KinPairData'
kinship(x)

## S4 replacement method for signature 'KinPairData'
kinship(x) <- value
```

Arguments

`x`	object of class `KinPairData`
`value`	value to assign to slot
`KinPairData`	object of class `KinPairData`

Value

returns `character` kinship category of object or `KinPairData` object with modified kinship category

Methods (by class)

- `kinship(KinPairData)`:
- `kinship(KinPairData) <- value`:

See Also

Other kpdmethods: `distances()`, `lifestage()`

kin_pair_data *Make new KinPairData object*

Description

Make new KinPairData object

Usage

```
kin_pair_data(data = NULL, kinship = NULL, lifestage = NULL, cycle = NULL)
```

Arguments

data	tlb_df. Tibble of kinpair distances
kinship	character. - one of PO, FS, HS, AV, HAV, GG, 1C, H1C, GAV, HGAV, 1C1, H1C1, GGG, 2C, H2C & UN.
lifestage	character. - one of 'unknown', 'immature' or 'ovipositional', or alternatively a custom stage that corresponds to a dispersal stage contained in a DispersalModel object.
cycle	non-negative integer of length one or two (here, 1 is equivalent to c(1, 1)). Represents the number of complete breeding cycles each individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). If the first individual was sampled as a juvenile & the second as an adult of equivalent stage, the vector c(0, 1) would be used. In most situations, the default will be appropriate

Value

returns an object of class KinPairData

Examples

```
kin_pair_data()
```

kin_pair_simulation *KinPairSimulation*

Description

KinPairSimulation

Usage

```
kin_pair_simulation(
  data = NULL,
  kinship = NULL,
  lifestage = NULL,
  simtype = NULL,
  kerneltype = NULL,
  posigma = NULL,
  initsigma = NULL,
  breedsigma = NULL,
  gravsigma = NULL,
  ovisigma = NULL,
  customsigma = NULL,
  simdims = NULL,
  kernelshape = NULL,
  cycle = NULL,
  call = NULL,
  filtertype = NULL,
  upper = NULL,
  lower = NULL,
  spacing = NULL,
  samplenum = NULL,
  sampledims = NULL,
  model = NULL
)
```

Arguments

data	tbl_df. tibble of simulation values
kinship	character - one of PO, FS, HS, AV, HAV, GG, 1C, H1C, GAV, HGAV, 1C1, H1C1, GGG, 2C, and H2C.
lifestage	character - one of 'unknown', 'immature' or 'ovipositional'
simtype	character - simulation type
kerneltype	character. - 'Gaussian', 'Laplace' or 'vgamma' (variance-gamma)
posigma	numeric - overall value of dispersal sigma (for simple kernel)
initsigma	numeric. - value of pre-breeding dispersal sigma (for composite kernel)
breedsigma	numeric. - value of breeding dispersal sigma (for composite kernel)
gravsigma	numeric. - value of post-breeding dispersal sigma (for composite kernel)
ovisigma	numeric. - value of oviposition dispersal sigma (for composite kernel)
customsigma	numeric. - vector of named custom dispersal sigmas (for custom kernel)
simdims	numeric. - dimensions of sampling area (assumes one side of square)
kernelshape	numeric. - value of kernel shape of simulation (if using kernel with shape parameter e.g. vgamma)
cycle	integer - number of breeding cycles sampled individual has survived (for custom kernel)

call	call. Call to create object
filtertype	character. whether the initial sim has been further filtered
upper	numeric. - FILTER: upper threshold used
lower	numeric. - FILTER: lower threshold used
spacing	numeric. - FILTER: spacing used
samplenum	numeric. - FILTER: sample number used
sampledims	numeric. - FILTER: sample dimensions used
model	list - model information if custom simulation used to generate object

Value

returns an object of class KinPairSimulation.

Examples

```
kin_pair_simulation()
```

---

lifestage *Access or assign* lifestage *category of* KinPairData *class objects*

Description

Access or assign lifestage category of KinPairData class objects

Usage

```
lifestage(x)

lifestage(x) <- value

## S4 method for signature 'KinPairData'
lifestage(x)

## S4 replacement method for signature 'KinPairData'
lifestage(x) <- value
```

Arguments

x	object with relevant method
value	new value to assign
KinPairData	object of class KinPairData

Value

returns character lifestage of object or KinPairData object with modified lifestage

Methods (by class)

- `lifestage(KinPairData)`:
- `lifestage(KinPairData) <- value`:

See Also

Other kpdmethods: `distances()`, `kinship()`

`mentari` *Position & kinship information of* Aedes aegypti *from Mentari Court, Malaysia*

Description

A data file containing the positions & kinship values of 98 *Ae. aegypti* larval kin pairs collected between September 19 & October 10, 2017 in Mentari Court (Petaling Jaya), Malaysia.

Usage

```
mentari
```

Format

A data frame with 98 rows and 10 variables

id1 id of first individual of kinpair

id2 id of second individual of kinpair

kinship kinship category of the pairing

distance geographical distance between kinpair

x1 relative x coordinate of first individual in metres

y1 relative y coordinate of first individual in metres

x2 relative x coordinate of second individual in metres

y2 relative y coordinate of second individual in metres

lifestage lifestage at time of sampling of kinpair

k_loiselle calculated Loiselle's *k* value for kinpair

Details

162 individuals were sourced as larvae from ovitraps placed in eight apartment buildings (in floors three or four for each), collected over three weeks. Entire larval bodies were extracted and sequenced using the double-digest restriction-site- associated DNA sequencing protocol for *Ae. aegypti* (doi:10.1186/1471216415275. After sequencing & genotyping, Loiselle's *k* was used as an initial estimate of genetic kinship. The program ML-Relate (doi:10.1111/j.14718286.2006.01256.x) was then used to estimate the pedigree kinships for the FS and HS categories. Following simulation work described in doi:10.1111/17550998.13043 the 1C category was assigned to all remaining unassigned individuals with a Loiselle's *k* of less than 0.06.

Value

returns an object of class `tbl_df`

Source

doi:10.1111/17550998.13043

`mount_appdata` *Mount* `KinPairData` *Objects for use in kindisperse app*

Description

This function is part of a suite of functions handling the interface between the kindisperse app & R package. Due to how shiny's interactive programming works, ordinary objects are not visible to the reactive functions embedded in the app. The solution implemented here is to construct a custom environment, `env_appdata`, that is accessible within the app and outside of it.

This function takes an object of class `KinPairData` or `KinPairSimulation`, assigns it an identifying name, and adds it to the app interface environment, making it accessible within the app. Once added, this object will be accessible under its name from the `Load` menu of the app. (The app interface uses the same function internally, enabling objects to be passed to the interface from the app also).

Usage

```
mount_appdata(x, nm)
```

Arguments

`x`	An object of class `KinPairData` or `KinPairSimulation`
`nm`	character. A name to store the object as

Value

invisibly returns x.

See Also

Other app_ports: `display_appdata()`, `reset_appdata()`, `reset_tempdata()`, `retrieve_appdata()`, `retrieve_tempdata()`, `retrieveall_appdata()`, `unmount_appdata()`

Examples

```
mount_appdata(kin_pair_data(), "mydata")
```

`read_kindata` *Reads* `.kindata` *filetype back to* `KinPairData` *or* `KinPairSimulation` *object.*

Description

This function is part of suite of functions handling file import/export for kinship dispersal objects.

The custom `.kindata` format enables complete preservation of `KinPairData` & `KinPairSimulation` formats without any loss of class attributes or metadata - ideal for saving and retrieving simulation data that is intended for further in-package processing with kindisperse. This function loads a previously stored object into its original class format.

Usage

```
read_kindata(file)
```

Arguments

`file`	Character giving path reference to file with extension `.kinpair`

Value

Returns either `KinPairData` or `KinPairSimulation` object.

See Also

Other import_functions: `csv_to_kinpair()`, `df_to_kinpair()`, `tsv_to_kinpair()`, `vector_to_kinpair()`

`rebase_dims` *Change the dimensions of a KinPairSimulation Object and shift kinpairs so at least one individual is within the area*

Description

Change the dimensions of a KinPairSimulation Object and shift kinpairs so at least one individual is within the area

Usage

```
rebase_dims(kindist, dims)
```

Arguments

`kindist`	`KinPairSimulation` - `KinPairSimulation` Class Object
`dims`	New site dimensions - either single number (length of side of square) or length 2 vector (lengths of sides x and y of rectangle)

Value

returns a rebased object of class KinPairSimulation with adjusted simulation dimensions

Examples

```
simobject <- simulate_kindist_simple()

rebase_dims(simobject, c(1, 100))
rebase_dims(simobject, 15)
```

---

reset_appdata *Reset kindisperse appdata*

Description

This function is part of a suite of functions handling the interface between the kindisperse app & R package. Due to how shiny's interactive programming works, ordinary objects are not visible to the reactive functions embedded in the app. The solution implemented here is to construct a custom environment, env_appdata, that is accessible within the app and outside of it.

When called, this function clears all attached objects from the app interface environment, keeping it from becoming over-clutttered & taking up space.

Usage

```
reset_appdata()
```

Value

No return value, called for side effects

See Also

Other app_ports: display_appdata(), mount_appdata(), reset_tempdata(), retrieve_appdata(), retrieve_tempdata(), retrieveall_appdata(), unmount_appdata()

Examples

```
reset_appdata()
```

reset_tempdata *Reset app tempdata (internal mem)*

Description

This function is part of a suite of functions handling the interface between the kindisperse app & R package. Due to how shiny's interactive programming works, ordinary objects are not visible to the reactive functions embedded in the app. The solution implemented here is to construct a custom environment, `env_appdata`, that is accessible within the app and outside of it.

This function resets the internal `tempdata` environment used by the kindisperse app, keeping it from becoming over-cluttered & freeing up space.

Usage

```
reset_tempdata()
```

Value

No return value, called for side effects

See Also

Other app_ports: `display_appdata()`, `mount_appdata()`, `reset_appdata()`, `retrieve_appdata()`, `retrieve_tempdata()`, `retrieveall_appdata()`, `unmount_appdata()`

Examples

```
reset_tempdata()
```

retrieveall_appdata *Retrieve all* `KinPairData` *objects from appdata (as list)*

Description

This function is part of a suite of functions handling the interface between the kindisperse app & R package. Due to how shiny's interactive programming works, ordinary objects are not visible to the reactive functions embedded in the app. The solution implemented here is to construct a custom environment, `env_appdata`, that is accessible within the app and outside of it.

This function accesses the app interface environment and retrieves a named list of all objects (typically of classes `KinPairData` or `KinPairSimulation` contained within it, making them accessible outside of the app). This is used to quickly pass all simulation objects that were saved to this interface environment while using the app to the regular R environment (after closing the app).

Usage

```
retrieveall_appdata()
```

Value

Returns a list of objects stored in the appdata environment

See Also

Other app_ports: display_appdata(), mount_appdata(), reset_appdata(), reset_tempdata(), retrieve_appdata(), retrieve_tempdata(), unmount_appdata()

Examples

```
mount_appdata(kin_pair_data(), "k1")
mount_appdata(kin_pair_simulation(), "s1")
retrieveall_appdata()
```

---

retrieve_appdata *Retrieve* KinPairData *object from appdata (single)*

Description

This function is part of a suite of functions handling the interface between the kindisperse app & R package. Due to how shiny's interactive programming works, ordinary objects are not visible to the reactive functions embedded in the app. The solution implemented here is to construct a custom environment, env_appdata, that is accessible within the app and outside of it.

This function accesses the app interface environment and retrieves an object (typically of class KinPairData or KinPairSimulation) with the name nm, making it accessible from within our outside the app. This can be used to load simulation objects that were saved from the interface while using the app into the regular R environment (after closing the app). (The app uses this function internally to load objects from the interface into its own internal environment for display & processing.)

Usage

```
retrieve_appdata(nm)
```

Arguments

nm	character. Name of item as stored in appdata

Value

Returns KinPairData object accessible by name nm

See Also

Other app_ports: display_appdata(), mount_appdata(), reset_appdata(), reset_tempdata(), retrieve_tempdata(), retrieveall_appdata(), unmount_appdata()

Examples

```
mount_appdata(kin_pair_data(), "mydata")

retrieve_appdata("mydata")
```

`retrieve_tempdata` *Retrieve all tempdata (internal mem) from app (as list)*

Description

This function is part of a suite of functions handling the interface between the kindisperse app & R package. Due to how shiny's interactive programming works, ordinary objects are not visible to the reactive functions embedded in the app. The solution implemented here is to construct a custom environment, `env_appdata`, that is accessible within the app and outside of it.

This function accesses the app internal environment and retrieves a named list of all objects (typically of classes `KinPairData` or `KinPairSimulation` contained within it, making them accessible outside of the app). This is used to quickly retrieve all objects stored in the app's internal memory. Ordinarily, these would be passed to the interface environment, but this function is useful if the app crashed and important results were only present in the app's internal environment.

Usage

```
retrieve_tempdata()
```

Value

A list of all KinPairData objects in kindisperse app's tempdata

See Also

Other app_ports: `display_appdata()`, `mount_appdata()`, `reset_appdata()`, `reset_tempdata()`, `retrieve_appdata()`, `retrieveall_appdata()`, `unmount_appdata()`

Examples

```
retrieve_tempdata()
```

run_kindisperse *Run kindisperse app*

Description

Run kindisperse app

Usage

```
run_kindisperse()
```

Value

returns a shiny app instance of kindisperse

sample_kindist *Subsample and filter a* KinPairSimulation *or* KinPairData *Object*

Description

This function takes a pre-existing KinPairSimulation or KinPairData Object with distance and coordinate data and filters it to simulate various in-field sampling schemes.

Usage

```
sample_kindist(
  kindist,
  upper = NULL,
  lower = NULL,
  spacing = NULL,
  n = NULL,
  dims = NULL
)
```

Arguments

kindist	KinPairSimulation Class Object
upper	numeric - upper cutoff for kin pair distances
lower	numeric - lower cutoff for kin pair distances
spacing	numeric - spacing between traps (location-independent)
n	numeric - number of individuals to keep after filtering (if possible)
dims	dimensions to sample within (works with the KinPairSimulation spatial & dimension information). Either num (defining a square) or c(num1, num2) (defining a rectangle).

Details

This function enables the testing of the impact of some basic sampling constraints that might be encountered in study design or implementation on the effectiveness of the `kindisperse` estimation of intergenerational dispersal. It is typically paired with a simulation function such as `simulate_kindist_composite` to generate a 'pure' dataset, then an estimation function such as `axpermute` to examine the impact of filter settings on the 'detected' value of dispersal sigma. The filter parameters `upper`, `lower`, & `spacing` all work on the vector of (direction-independent) distances, & the parameter `n` enables the random subsampling of n kin dyads. The parameter `dims` requires 2D location information for each individual, meaning it can ordinarily only be used with the `KinPairSimulation` object (not `KinPairData`). All filter parameters are stackable.

The `upper` parameter implements a cutoff for the **maximum** distance allowable in the dataset. If set to e.g. 100m, all kin dyads separated by a distance greater than 100m will be excluded from the filtered dataset. Note that this is a geometry-independent metric; it is naive to the edge effects of an actual sample site. The `lower` parameter implements a cutoff for the **minimum** distance allowable in the dataset.It operates in the same manner as the previous parameter (in this case, removing results smaller than a distance threshold)

The `spacing` parameter as currently implemented takes all distances & alters them to lie at the midpoint of a bin with width set by this parameter. So if `spacing` is set to 10 meters, all kin pairs with distances between 0 and 10m will have their distances rest to 5m, all between 10 & 20 will be set to 15 m, etc. (quantizing the data). Note that once again this is a geometry-independent action: These binwiths & 'trap spacing' are not spatially related to each other like they would be in a sample site, and there is no simulated dropout of kinpairs too far from a trap. There is also no geometry-dependent profiling of possible frequency of recaptures across each distance category (will be implemented in a future version). (this parameter leaves 2D spatial information intact)

The `dims` parameter defines the dimensions of a rectangle within which **both** individuals of a kin dyad will need to lie to be included in the filtered dataset. This measure (which excludes e.g. long-distance dispersal into & out of the study site) is *geometry-dependent*, unlike the `upper` parameter. This enables the testing of (rectangular) site geometries potentially corresponding to an actual site (two-dimensional estimates of dispersal such as `kindisperse` become unreliable as edge effects significantly reduce the size of either one or both dimensions with respect to the real underlying dispersal sigma). These site geometries can be entered in a few ways: (a) a single numeric value, which will be interpreted as the length of the side of a square; (b) a numeric vector of length two, which will be interpreted as the length & width of the sample site; (c) either of the above passed to the `elongate` function, which takes the rectangular site dimensions and alters their `aspect ratio` (ratio of length to width) while preserving the underlying area the study site covers. The implementation of this filtering step permutes the absolute positions of all dyads so that at least one member of the dyad is in the inial site rectangle, while preserving their relative positions (and angles) with respect to each other. This means that following this step, the xy coordinate positions of each individual will not match those contained in the previous round. It also means that the repeated calling of this function will result in a steady reduction in retained kin dyads due to edge effects.

The `n` parameter randomly samples `n` pairs from the dataset. It is implemented after all other filtering has taken place, so will only sample surviving individuals A typical strategy for the use of this functions in simulations would be to simulate an extremely large (e.g. one million pairs) dataset, then pass it repeatedly to this filter function, with a final sub-sampling step of 1,000 included. This enables comparisons across sampling conditions (in most cases) regardless of the amount of data filtered prior to this step.

As this function returns a KinPairData or KinPairSimulation object, the returned object can be passed back for filtering an arbitrary number of times, or alternatively passed to an estimation strategy.

This function can be used to test for bias in the results of a close-kin dispersal study that has been conducted. After the field sampling, kin identification, & sigma calculation steps, use the estimated sigmas as inputs into simulation functions that are then filtered for size & geometry of the actual study site (via the dims method). Then pass this filtered dataset back to the sigma-determining functions. If filtering has resulted in a substantial drop in sigma, the estimate of sigma from the study site has likely been biased by the site geometry (note that the impact of this is dependent on the shape of the dispersal kernel - the more leptokurtic (dominated by long-distance dispersal), the more severe bias will be for a particular sigma and site geometry.

Value

returns an object of class KinPairData or KinPairSimulation containing simulation and filtering details and a filtered dataset of dispersed individuals.

Examples

```
simobject <- simulate_kindist_simple(nsims = 100000, sigma = 100, kinship = "PO")

sample_kindist(simobject, upper = 200, lower = 50, spacing = 15, n = 100)
```

---

sampling_stage *Access sampling stage of* DispersalModel *or* KinPairSimulation *object.*

Description

Access sampling stage of DispersalModel or KinPairSimulation object.

Usage

```
sampling_stage(x)

sampling_stage(x) <- value

## S4 method for signature 'DispersalModel'
sampling_stage(x)

## S4 replacement method for signature 'DispersalModel'
sampling_stage(x) <- value

## S4 method for signature 'KinPairData'
sampling_stage(x)
```

Arguments

x	object of class DispersalModel
value	character new sampling stage to assign model
DispersalModel	object of class DispersalModel
KinPairData	object of class KinPairData

Value

character sampling stage

returns a modified object of class DispersalModel

Methods (by class)

- sampling_stage(DispersalModel):
- sampling_stage(KinPairData):

---

simdims *Access simulation dimensions of* KinPairSimulation *object*

Description

Access simulation dimensions of KinPairSimulation object

Usage

```
simdims(x)

simdims(x) <- value

## S4 method for signature 'KinPairSimulation'
simdims(x)
```

Arguments

x	object of class KinPairSimulation
value	new value to assign
KinPairSimulation	object of class KinPairSimulation

Value

numeric vector dimensions of simulated object

returns a modified object of the relevant class

numeric vector simulation dimensions of KinPairSimulation object

Methods (by class)

- simdims(KinPairSimulation):

---

simgraph_data *Simple kin dispersal simulation for graphical display. (returns the data side as a tibble).*

Description

Simple kin dispersal simulation for graphical display. (returns the data side as a tibble).

Usage

```
simgraph_data(nsims = 1000, posigma = 50, dims = 250, kinship = "2C")
```

Arguments

nsims	Integer. The number of kin dispersal families to simulate.
posigma	Integer. The axial deviation of the (simple) parent-offspring dispersal kernel governing this simulation.
dims	Integer. Lays out the length of the sides of a square within which parent individuals are seeded.
kinship	Character. Lists the kin category the simulation is reconstructing. One of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV" (no half-categories included)

Value

Returns a tibble containing the coordinates of the f0 to f2 generations, as well as coordinates and distances relative to the 'focus' kinship categories. (kindist, kinmid, k1 & k2)

See Also

Other simgraph: simgraph_graph()

Examples

```
simgraph_data(nsims = 100, dims = 1000, kinship = "GAV")
```

simgraph_graph *Simple kin dispersal simulation for graphical display. (graphs the pre-existing simulation).*

Description

Simple kin dispersal simulation for graphical display. (graphs the pre-existing simulation).

Usage

```
simgraph_graph(
  result,
  nsims = 10,
  labls = TRUE,
  steps = TRUE,
  moves = TRUE,
  shadows = TRUE,
  kinship = NULL,
  show_area = TRUE,
  centred = FALSE,
  pinwheel = FALSE,
  scattered = FALSE,
  lengths = TRUE,
  lengthlabs = TRUE,
  histogram = FALSE,
  binwidth = posigma/5,
  freqpoly = FALSE
)
```

Arguments

result	simulation supplied from simgraph_data() function (tibble)
nsims	number of families to graph
labls	Logical. Displays labels.
steps	Logical. Whether or not to show any details of dispersal movement
moves	Logical. Whether or not to show (curved) lines denoting dispersal movement
shadows	Logical. Whether or not to show (dashed) shadows tracing dispersal movement.
kinship	Character. Lists the kin category the simulation is reconstructing. One of "PO", "FS", "HS", "AV", "GG", "HAV", "GGG", "1C", "1C1", "2C", "GAV" (no half-categoris included)
show_area	Logical. Whether or not to show the parental seed area as defined in data$dims
centred	Logical. Whether or not to centre the coordinates on one individual.
pinwheel	Logical. Whether the final graph should be of the pinwheel form.
scattered	Logical. Whether the final graph should be of the scatter form.

lengths	Logical. Whether or not to show a dashed line connecting the 'focus' kin to illustrate overall distance of dispersal.
lengthlabs	Logical. Whether to show labels denoting distance of dispersal between focus kin.
histogram	Logical. Whether the final graph should be of the histogram form.
binwidth	Numeric. Binwidth for histogram or freqpoly.
freqpoly	Logical. Whether the final graph should be of the freqpoly form.

Value

Returns a ggplot object for graphing.

See Also

Other simgraph: simgraph_data()

Examples

```
simdata <- simgraph_data()
simgraph_graph(simdata)
```

---

simtype *Access or assign simulation type of* KinPairSimulation *object*

Description

Access or assign simulation type of KinPairSimulation object

Usage

```
simtype(x)

simtype(x) <- value

## S4 method for signature 'KinPairSimulation'
simtype(x)
```

Arguments

x	object of class KinPairSimulation
value	new value to assign
KinPairSimulation	object of class KinPairSimulation

Value

`character` the kind of simulation stored in the object (simple or composite)

returns a modified object of relevant class

`character` the kind of simulation stored in the object (simple or composite)

Methods (by class)

- `simtype(KinPairSimulation)`:

See Also

Other kpsmethods: `access_sigmas`, `filter_methods`, `kernelshape()`, `kerneltype()`

`simulate_kindist_composite` *Simulate kin dispersal distance pairs with composite sigmas*

Description

Simulates intergenerational dispersal made up of composite dispersal stages in a species with a defined breeding and dispersal structure similar to that of *Ae. aegypti* - i.e. with initial, breeding, gravid & ovipositional dispersal phases, approximately non-overlapping life cycles, and defined sampling points.

Usage

```
simulate_kindist_composite(
  nsims = 100,
  initsigma = 100,
  breedsigma = 50,
  gravsigma = 50,
  ovisigma = 25,
  dims = 100,
  method = "Gaussian",
  kinship = "FS",
  lifestage = "immature",
  shape = 0.5
)
```

Arguments

`nsims`	(integer) - number of pairs to simulate
`initsigma`	(numeric) - size of pre-breeding (axial) sigma
`breedsigma`	(numeric) - size of breeding (axial) sigma
`gravsigma`	(numeric) - size of post-breeding (axial) sigma

ovisigma	(numeric) - size of oviposition (axial) sigma
dims	(numeric) - length of sides of (square) simulated site area
method	(character) - kernel shape to use: either 'Gaussian', 'Laplace' or 'vgamma' (variance-gamma)
kinship	(character)- kin category to simulate: one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C H1C1 or H2C
lifestage	(character) lifestage at sample collection: either 'immature' or 'ovipositional'
shape	(numeric) - value of shape parameter to use with 'vgamma' method. Default 0.5. Must be > 0. Increment towards zero for increasingly heavy-tailed (leptokurtic) dispersal

Details

This function is one of a family of functions that implement the core intergenerational dispersal simulations contained in the kindisperse package. Each of these functions proceeds by the following steps:

1. identify the pedigree relationship, dispersal phase (FS, HS & PO) and sampling stage that must be generated;
2. randomly assign a coordinate position to the 'root' individual within the pedigree (i.e. last common ancestor of the dyad, inclusive);
3. 'disperse' both pathways from this root position via the appropriately defined phase dispersal (additively via random draws from the underlying statistical model, defined by an axial standard deviation - sigma);
4. further disperse both phased descendant branches according to the number of realised breeding dispersal cycles contained in the defining pedigree (additively via random draws from the chosen underlying statistical model);
5. add displacement caused by dispersal before the sampling point in a similar manner to above, defining the final positions of the sampled dispersed kin dyads;
6. calculating geographical distances between the resulting dyads.

These simulation functions operate under an additive variance framework: all individual dispersal events are modeled as random draws from a bivariate probability distribution defined by an axial standard deviation sigma and (sometimes) a shape parameter. At present, three such distributions are included as options accessible with the method parameter: the bivariate normal distribution 'Gaussian', the bivariate Laplace distribution 'Laplace', and the bivariate variance-gamma distribution 'vgamma'. The Gaussian (normal) distribution enables easy compatibility with the framework under which much population genetic & dispersal theory (isolation by distance, neighbourhoods, etc.) have been developed. The Laplace distribution is a multivariate adaptation of the (positive) exponential distribution, and represents a more 'fat-tailed' (leptokurtic) disperal situation than Gaussian. The vgamma distribution is a mixture distribution formed by mixing the gamma distribution with the bivariate normal distribution. The flexibility of this distribution's shape parameter enables us to model arbitrarily leptokurtic dispesal kernels, providing a helpful way to examine the impacts of (e.g.) long distance dispersal on the overall disperal distribution and sampling decisions. A vgamma distribution with shape parameter equal to 1 reduces to the bivariate Laplace distribution. As shape approaches infinity, the vgamma distribution approaches the bivariate normal distribution. As shape approaches zero, the distribution becomes increasingly leptokurtic.

The simulate_kindist_composite() function is designed to enable modeling of the composite dispersal events that occur **within** the breeding cycle of an organism, and enables the separate treatment of the PO, FS, and HS phases (where, for example, the final distributions of full and half siblings are different in contexts where males mate with multiple females but females primarily carry the offspring of one male). This function has been designed primarily in the context of modelling dispersal in the mosquito *Ae. aegypti*; parameter names and the structure of kinship phases reflect a single-generational breeding organism with an initial dispersal phase, a mating phase (where HS individuals branch), a gravid phase, and an oviposition phase (where FS individuals branch). The sampling options ('immature' & 'ovipositional') also reflect common mosquito trapping methods (i.e. ovitraps & gravitraps) which both target individuals dispersing in the defined oviposition phase. This function should be easily adaptable to a vast number of other animals, especially insects, where breeding occurs in one generation and parameters such as this hold. For slightly more complex scenarios (multiple breeding cycles, differing sample points, more or less dispersal components making up a lifespan, different FS/HS branchpoints, etc.), the enhanced capabilities of the simulate_kindist_custom function may be required.

Following simulation, the results are returned as an object of the specially defined package class KinPairSimulation, which stores the simulation results along with information about all simulation parameters, and can be further passed to sample filtering & dispersal estimation functions.

Value

returns an object of class KinPairSimulation containing simulation details and a tibble (tab) of simulation values

See Also

Other simulate_kindist: simulate_kindist_custom(), simulate_kindist_simple()

Examples

```
simulate_kindist_composite(nsims = 100)
simulate_kindist_composite(
  nsims = 10000, initsigma = 20, breedsigma = 30, gravsigma = 30,
  ovisigma = 12, dims = 500, method = "Laplace", kinship = "1C", lifestage = "immature"
)
```

---

simulate_kindist_custom

Simulate kin dispersal distance pairs with custom species dispersal models.

Description

Simulates intergenerational dispersal in a species defined by multiple dispersal components across the breeding cycle, with dispersal, breeding & sampling & basic generational structure custom-defined by a DispersalModel object.

Usage

```
simulate_kindist_custom(
  nsims = 100,
 model = dispersal_model(init = 100, breed = 50, grav = 50, ovi = 25, .FS = "ovi", .HS =
    "breed"),
  dims = 100,
  method = "Gaussian",
  kinship = "FS",
  cycle = 0,
  shape = 0.5
)
```

Arguments

nsims	(integer) - number of pairs to simulate
model	(object of class DispersalModel) - species-specific model of dispersal containing lifestage, phase & sampling parameters
dims	(numeric) - length of sides of (square) simulated site area
method	(character) - kernel shape to use: either 'Gaussian', 'Laplace' or 'vgamma' (variance-gamma)
kinship	(character)- kin category to simulate: one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C H1C1 or H2C
cycle	(numeric) - breeding cycle number(s) of dispersed kin to be modeled. Must be a integer equal to or greater than -1, (-1, 0, 1, 2, ...) or vector of two such integers. Represents the number of complete breeding cycles each simulated individual has undergone before the sampling point, where the time between birth and first reproduction is coded as '0', that between first and second reproduction '1', etc. (default 0). If cycle is specially set to '-1' this constitutes the sampling of an individual before it has differentiated (via dispersal) from the parent. Only use in spp. where there is likely to be a reasonable equivalence between breeding stages across a lifespan.
shape	(numeric) - value of shape parameter to use with 'vgamma' method. Default 0.5. Must be > 0. Increment towards zero for increasingly heavy-tailed (leptokurtic) dispersal

Details

This function is one of a family of functions that implement the core intergenerational dispersal simulations contained in the kindisperse package. Each of these functions proceeds by the following steps:

1. identify the pedigree relationship, dispersal phase (FS, HS & PO) and sampling stage that must be generated;
2. randomly assign a coordinate position to the 'root' individual within the pedigree (i.e. last common ancestor of the dyad, inclusive);

3. 'disperse' both pathways from this root position via the appropriately defined phase dispersal (additively via random draws from the underlying statistical model, defined by an axial standard deviation - sigma);
4. further disperse both phased descendant branches according to the number of realised breeding dispersal cycles contained in the defining pedigree (additively via random draws from the chosen underlying statistical model);
5. add displacement caused by dispersal before the sampling point in a similar manner to above, defining the final positions of the sampled dispersed kin dyads;
6. calculating geographical distances between the resulting dyads.

These simulation functions operate under an additive variance framework: all individual dispersal events are modeled as random draws from a bivariate probability distribution defined by an axial standard deviation `sigma` and (sometimes) a shape parameter. At present, three such distributions are included as options accessible with the `method` parameter: the bivariate normal distribution '`Gaussian`', the bivariate Laplace distribution '`Laplace`', and the bivariate variance-gamma distribution '`vgamma`'. The `Gaussian` (normal) distribution enables easy compatibility with the framework under which much population genetic & dispersal theory (isolation by distance, neighbourhoods, etc.) have been developed. The `Laplace` distribution is a multivariate adaptation of the (positive) exponential distribution, and represents a more 'fat-tailed' (leptokurtic) disperal situation than Gaussian. The `vgamma` distribution is a mixture distribution formed by mixing the gamma distribution with the bivariate normal distribution. The flexibility of this distribution's `shape` parameter enables us to model arbitrarily leptokurtic dispesal kernels, providing a helpful way to examine the impacts of (e.g.) long distance dispersal on the overall disperal distribution and sampling decisions. A `vgamma` distribution with shape parameter equal to 1 reduces to the bivariate Laplace distribution. As shape approaches infinity, the `vgamma` distribution approaches the bivariate normal distribution. As shape approaches zero, the distribution becomes increasingly leptokurtic.

The `simulate_kindist_custom()` function is designed to enable modeling of the composite dispersal events that occur **within** the breeding cycle of an organism, and enables the separate treatment of the `PO`, `FS`, and `HS` phases in situations where the breeding and dispersal cycle of an organism is (somewhat more complex that that encountered in organisms such as mosquitoes (i.e. single-generational breeding organisms with defined sampling points). This function relies on a custom dispersal model of class `DispersalModel` defined via parameter `model` to supply organism-specific information about dispersal stages (with axial sigmas), FS & HS branch points, and the dispersal stage at which sampling occurs. Via this model object (or overridden by the `cycle` parameter) you can also define the number of breeding cycles each final individual within the close-kin dyad has passed through before sampling. This is defined as a length one or two non-negative integer (where a length-one integer of value a is converted to a length two integer of value c(a, a)), where the first integer defines the number of life cycles passed through by the 'senior' pedigree member of the dyad, and the second the number passed through by the 'junior' member (so the GG phase has a grandparent as senior, the grandchild as junior, etc. (in practice this distinction is unimportant for dyads). A cycle number of 0 references an individual that hasn't lived through an entire breeding cycle (sampling phase to sampling phase) before being sampled. A value of 1 references an individual that has lived through one such cycle (e.g. a female entering her second breeding season, an ovipositing mosquito (where the oviposition dispersal stage overlaps with the larval dispersal stage)). A value of 2 references two such cycles, etc. As all cycles are considered equivalent in the current formulation of this model (whether an individual enters the cycle as a juvenile or as an adult) care must be taken in applying this system to species where the dispersal behaviour of

a second cycle individual (i.e. adult) is likely to be substantially different to that of a first cycle individual (often an immature individual).

This function can only handle one kinship pairing & dispersal mode in the one simulation: where multiple dispersal pathways lead to the same kinship outcome, each pathway should be simulated separately, and the resulting distributions combined subsequently.

Following simulation, the results are returned as an object of the specially defined package class KinPairSimulation, which stores the simulation results along with information about all simulation parameters, and can be further passed to sample filtering & dispersal estimation functions.

Value

returns an object of class KinPairSimulation containing simulation details and a tibble (tab) of simulation values

See Also

Other simulate_kindist: simulate_kindist_composite(), simulate_kindist_simple()

Examples

```
custom_dispersal_model <- dispersal_model(a = 10, b = 25, .FS = "b",
.HS = "a", .sampling_stage = "b")
simulate_kindist_custom(nsims = 100, model = custom_dispersal_model,
cycle = c(0, 1), kinship = "FS")
```

---

simulate_kindist_simple

Simulate kin dispersal distance pairs with simple sigma

Description

Simulates intergenerational dispersal defined by a simple dispersal sigma (covering the entire life-cycle) and ignoring phase differences between full & half sibling dispersal categories. Returns an object of class KinPairSimulation

Usage

```
simulate_kindist_simple(
  nsims = 100,
  sigma = 125,
  dims = 100,
  method = "Gaussian",
  kinship = "PO",
  lifestage = "immature",
  shape = 0.5
)
```

Arguments

nsims	(integer) - number of pairs to simulate
sigma	(numeric) - size of simple (axial) sigma
dims	(numeric) - length of sides of (square) simulated site area
method	(character) - kernel shape to use: either 'Gaussian', 'Laplace' or 'vgamma' (variance-gamma)
kinship	(character)- kin category to simulate: one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C or H2C
lifestage	(lifestage) lifestage at sample collection: either 'immature' or 'ovipositional'
shape	(numeric) - value of shape parameter to use with 'vgamma' method. Default 0.5. Must be > 0. Increment towards zero for increasingly heavy-tailed (leptokurtic) dispersal

Details

This function is one of a family of functions that implement the core intergenerational dispersal simulations contained in the kindisperse package. Each of these functions proceeds by the following steps:

1. identify the pedigree relationship, dispersal phase (FS, HS & PO) and sampling stage that must be generated;
2. randomly assign a coordinate position to the 'root' individual within the pedigree (i.e. last common ancestor of the dyad, inclusive);
3. 'disperse' both pathways from this root position via the appropriately defined phase dispersal (additively via random draws from the underlying statistical model, defined by an axial standard deviation - sigma);
4. further disperse both phased descendant branches according to the number of realised breeding dispersal cycles contained in the defining pedigree (additively via random draws from the chosen underlying statistical model);
5. add displacement caused by dispersal before the sampling point in a similar manner to above, defining the final positions of the sampled dispersed kin dyads;
6. calculating geographical distances between the resulting dyads.

These simulation functions operate under an additive variance framework: all individual dispersal events are modeled as random draws from a bivariate probability distribution defined by an axial standard deviation sigma and (sometimes) a shape parameter. At present, three such distributions are included as options accessible with the method parameter: the bivariate normal distribution 'Gaussian', the bivariate Laplace distribution 'Laplace', and the bivariate variance-gamma distribution 'vgamma'. The Gaussian (normal) distribution enables easy compatibility with the framework under which much population genetic & dispersal theory (isolation by distance, neighbourhoods, etc.) have been developed. The Laplace distribution is a multivariate adaptation of the (positive) exponential distribution, and represents a more 'fat-tailed' (leptokurtic) disperal situation than Gaussian. The vgamma distribution is a mixture distribution formed by mixing the gamma distribution with the bivariate normal distribution. The flexibility of this distribution's shape parameter enables us to model arbitrarily leptokurtic dispesal kernels, providing a helpful way to examine the

impacts of (e.g.) long distance dispersal on the overall disperal distribution and sampling decisions. A vgamma distribution with shape parameter equal to 1 reduces to the bivariate Laplace distribution. As shape approaches infinity, the vgamma distribution approaches the bivariate normal distribution. As shape approaches zero, the distribution becomes increasingly leptokurtic.

The simulate_kindist_simple() function is the most basic of the simulation functions, ignoring all information about dispersal phase and treating dispersal with a single sigma corresponding to the entire lifecycle to breeding of the dispersed individuals. It is useful for exploring simple intergenerational dispersal in a stripped back context; for many typical contexts involving complex dispersal across different phases of the breeding cycle, the other dispersal simulation functions would be more suitable.

Following simulation, the results are returned as an object of the specially defined package class KinPairSimulation, which stores the simulation results along with information about all simulation parameters, and can be further passed to sample filtering & dispersal estimation functions.

Value

returns an object of class \code{\link{KinPairSimulation}} containing simulation details and a \code{t

See Also

Other simulate_kindist: simulate_kindist_composite(), simulate_kindist_custom()

Examples

```
test <- simulate_kindist_simple(nsims = 10, sigma = 50, dims = 1000, method = "Laplace")
simulate_kindist_simple(nsims = 10000, sigma = 75, kinship = "PO", lifestage = "ovipositional")
```

---

stages *Access breeding cycle stages of* DispersalModel *object.*

Description

Access breeding cycle stages of DispersalModel object.

Usage

```
stages(x)

## S4 method for signature 'DispersalModel'
stages(x)
```

Arguments

x	object of class DispersalModel
DispersalModel	object of class DispersalModel

Value

`character` ordered vector of custom lifestages contained in the object

Methods (by class)

- `stages(DispersalModel)`:

`tsv_to_kinpair` *Reads* `.tsv` *and converts to* `KinPairData` *object*

Description

This function is part of suite of functions handling file import/export for kinship dispersal objects.

`.csv` & `.tsv` reading functions at minimum require the .delim file to contain a column titled 'distance' containing distances between kin pairs. It can optionally contain a column of kinship values 'kinship' as well as a column of lifestage values 'lifestage'. If the file contains more than one value in the kinship or lifestage columns (e.g. bot 'FS' and 'HS') - the corresponding function parameter must be set to pick a corresponding subset of dispersed pairs. where parameters are set in the absence of file columns, these values are assigned to the returned `KinPairData` object.

Usage

```
tsv_to_kinpair(file, kinship = NULL, lifestage = NULL, ...)
```

Arguments

`file`	The file path to read from
`kinship`	character. kin category to assign or extract from data. one of PO, FS, HS, AV, GG, HAV, GGG, 1C, 1C1, 2C, GAV, HGAV, H1C , H1C1 or H2C
`lifestage`	character. lifestage to assign or extract from data. one of 'unknown', 'immature' or 'ovipositional'.
`...`	additional arguments to pass to `read_tsv`

Value

Returns object of class `KinPairData`

See Also

Other import_functions: `csv_to_kinpair()`, `df_to_kinpair()`, `read_kindata()`, `vector_to_kinpair()`

unmount_appdata *Unmount a* KinPairData *Object (clear slot from appdata environment)*

Description

This function is part of a suite of functions handling the interface between the kindisperse app & R package. Due to how shiny's interactive programming works, ordinary objects are not visible to the reactive functions embedded in the app. The solution implemented here is to construct a custom environment, env_appdata, that is accessible within the app and outside of it.

When called, this function clears any objects with names found in the vector nms from the app interface environment, keeping it from becoming over-cluttered & taking up space.

Usage

```
unmount_appdata(nms)
```

Arguments

nms	A character vector of names of objects to unmount from the appdata environment

Value

No return value, called for side effects

See Also

Other app_ports: display_appdata(), mount_appdata(), reset_appdata(), reset_tempdata(), retrieve_appdata(), retrieve_tempdata(), retrieveall_appdata()

Examples

```
mount_appdata(kin_pair_data(), "mydata")

unmount_appdata("mydata")
```

vector_to_kinpair *Convert vector of kin separation distances to* KinPairData *class*

Description

Function takes at minimum a (numeric) vector of distances between related kinpairs, and returns a KinPairData object. Optional parameters can assign kinship and lifestage values to the returned object.

Usage

```
vector_to_kinpair(vect, kinship = NULL, lifestage = NULL)
```

Arguments

vect	vector of kinpair distances
kinship	character or character vector containing kinship categories of kinpairs
lifestage	character or character vector containing lifestages of kinpairs

Value

returns valid KinPairData object.

See Also

Other import_functions: csv_to_kinpair(), df_to_kinpair(), read_kindata(), tsv_to_kinpair()

Examples

```
vector_to_kinpair(1:10, "FS", "immature")
```

visible_stage *Access life stage at which individual is first visible to sampling (i.e. from which breeding cycles are calculated)*

Description

Access life stage at which individual is first visible to sampling (i.e. from which breeding cycles are calculated)

Usage

```
visible_stage(x)

## S4 method for signature 'DispersalModel'
visible_stage(x)
```

Arguments

x	object of class DispersalModel or
DispersalModel	object of class DispersalModel

Value

character stage in life cycle at which an individual is assumed to be sampled by default rather than its parent (anchors the breeding cycle system)

Methods (by class)

- visible_stage(DispersalModel):

---

write_kindata *Write* KinPairData *or* KinPairSimulation *object in* .kindata *format*

Description

This function is part of suite of functions handling file import/export for kinship dispersal objects. Writing to the custom .kindata format enables complete preservation of KinPairData & KinPairSimulation formats without any loss of class attributes or metadata - ideal for saving simulation data that is intended for further in-package processing with kindisperse.

Usage

```
write_kindata(x, file)
```

Arguments

x	Object of class KinPairData or KinPairSimulation
file	The file path to write to. If is doesn't end it '.kindata', this will be added.

Value

Invisibly returns the initial object

See Also

Other export_functions: kinpair_to_csv(), kinpair_to_tibble(), kinpair_to_tsv()

Index

∗ **app_ports**
  display_appdata, 30
  mount_appdata, 55
  reset_appdata, 57
  reset_tempdata, 58
  retrieve_appdata, 59
  retrieve_tempdata, 60
  retrieveall_appdata, 58
  unmount_appdata, 77
∗ **axial_helpers**
  axials, 5
  axials_add, 6
  axials_decompose, 7
  axials_subtract, 12
  axpermute, 13
  axpermute_subtract, 18
∗ **axstandard**
  axials_standard, 8
  axpermute_standard, 14
∗ **datasets**
  mentari, 54
∗ **export_functions**
  kinpair_to_csv, 48
  kinpair_to_tibble, 49
  kinpair_to_tsv, 49
  write_kindata, 79
∗ **import_functions**
  csv_to_kinpair, 21
  df_to_kinpair, 22
  read_kindata, 56
  tsv_to_kinpair, 76
  vector_to_kinpair, 78
∗ **kdclasses**
  DispersalModel-class, 23
  KinPairData-class, 39
  KinPairSimulation-class, 41
∗ **kpdmethods**
  distances, 30
  kinship, 50
  lifestage, 53
∗ **kpsmethods**
  access_sigmas, 3
  filter_methods, 32
  kernelshape, 38
  kerneltype, 38
  simtype, 67
∗ **simgraph**
  simgraph_data, 65
  simgraph_graph, 66
∗ **simulate_kindist**
  simulate_kindist_composite, 68
  simulate_kindist_custom, 70
  simulate_kindist_simple, 73

access_sigmas, 3, *34*, *38, 39*, *68*
axials, 5, *6*, *8*, *12, 13*, *19*
axials_add, *5*, 6, *8*, *12, 13*, *19*
axials_combine, 7
axials_decompose, *5, 6*, 7, *12, 13*, *19*
axials_standard, 8, *16, 17*, *40*, *43*
axials_subtract, *5, 6*, *8*, 12, *13*, *19*
axpermute, *5, 6*, *8*, *12*, 13, *19*, *62*
axpermute_standard, *11, 12*, 14, *40*, *43*
axpermute_subtract, *5, 6*, *8*, *12, 13*, 18

breeding_cycle, 19, *25*, *28*, *40*, *43*
breeding_cycle,DispersalModel-method *(*breeding_cycle*)*, 19
breeding_cycle,KinPairData-method *(*breeding_cycle*)*, 19
breeding_stage, 20
breeding_stage,DispersalModel-method *(*breeding_stage*)*, 20
breedsigma *(*access_sigmas*)*, 3
breedsigma,KinPairSimulation-method *(*access_sigmas*)*, 3
breedsigma<- *(*access_sigmas*)*, 3

check_valid_kinship, 20

check_valid_lifestage, 21
csv_to_kinpair, 21, *22*, *40*, *56*, *76*, *78*

df_to_kinpair, *22*, 22, *40*, *56*, *76*, *78*
dispersal_model, 26
dispersal_vector, *24*, *28*, 29
dispersal_vector,DispersalModel-method *(dispersal_vector)*, 29
DispersalModel, *19, 20*, *28, 29*, *34–36*, *63*, *70*, *72*, *75*
DispersalModel *(DispersalModel-class)*, 23
DispersalModel-class, 23
display_appdata, 30, *55*, *57–60*, *77*
distances, 30, *40*, *43*, *50*, *54*
distances,KinPairData-method *(distances)*, 30

elongate, 31, *62*

filter_methods, *5*, 32, *38, 39*, *68*
filtertype, 32
filtertype,KinPairSimulation-method *(filtertype)*, 32
filtertype<- *(filtertype)*, 32
fs, *25*, 34
fs,DispersalModel-method *(fs)*, 34

get_dispersal_model, 35
get_dispersal_model,KinPairSimulation-method *(get_dispersal_model)*, 35
gravsigma *(access_sigmas)*, 3
gravsigma,KinPairSimulation-method *(access_sigmas)*, 3
gravsigma<- *(access_sigmas)*, 3

hs, *25*, 36
hs,DispersalModel-method *(hs)*, 36

initialize,DispersalModel-method *(DispersalModel-class)*, 23
initialize,KinPairData-method *(KinPairData-class)*, 39
initialize,KinPairSimulation-method *(KinPairSimulation-class)*, 41
initsigma *(access_sigmas)*, 3
initsigma,KinPairSimulation-method *(access_sigmas)*, 3
initsigma<- *(access_sigmas)*, 3
is.DispersalModel, 36
is.KinPairData, 37
is.KinPairSimulation, 37

kernelshape, *5*, *34*, 38, *39*, *68*
kernelshape,KinPairSimulation-method *(kernelshape)*, 38
kerneltype, *5*, *34*, *38*, 38, *68*
kerneltype,KinPairSimulation-method *(kerneltype)*, 38
kerneltype<- *(kerneltype)*, 38
kin_pair_data, 51
kin_pair_simulation, 51
kinpair_to_csv, 48, *49, 50*, *79*
kinpair_to_tibble, *49*, 49, *50*, *79*
kinpair_to_tsv, *49*, 49, *79*
KinPairData, *5*, *8*, *11*, *14*, *17*, *22*, *30*, *32*, *41*, *50*, *53*, *61*, *78*
KinPairData *(KinPairData-class)*, 39
KinPairData-class, 39
KinPairSimulation, *3*, *8*, *11*, *14*, *17*, *32*, *34, 35*, *38*, *57*, *61*, *63, 64*, *67*, *70*, *73*, *75*
KinPairSimulation *(KinPairSimulation-class)*, 41
KinPairSimulation-class, 41
KinPairSimulation_composite, 44
KinPairSimulation_custom, 45
KinPairSimulation_simple, 47
kinship, *31*, *40*, *43*, 50, *54*
kinship,KinPairData-method *(kinship)*, 50
kinship<- *(kinship)*, 50
kinship<-,KinPairData-method *(kinship)*, 50

lifestage, *31*, *40*, *43*, *50*, 53
lifestage,KinPairData-method *(lifestage)*, 53
lifestage<- *(lifestage)*, 53
lifestage<-,KinPairData-method *(lifestage)*, 53
lower *(filter_methods)*, 32
lower,KinPairSimulation-method *(filter_methods)*, 32
lower<- *(filter_methods)*, 32
lower<-,KinPairSimulation-method *(filter_methods)*, 32

mentari, 54
mount_appdata, *30*, 55, *57–60*, *77*

ovisigma *(access_sigmas)*, 3

ovisigma,KinPairSimulation-method
(access_sigmas), 3
ovisigma<- (access_sigmas), 3

posigma (access_sigmas), 3
posigma,KinPairSimulation-method
(access_sigmas), 3
posigma<- (access_sigmas), 3

read_kindata, *22*, 56, *76*, *78*
rebase_dims, 56
reset_appdata, *30*, *55*, 57, *58–60*, *77*
reset_tempdata, *30*, *55*, *57*, 58, *59, 60*, *77*
retrieve_appdata, *30*, *55*, *57–59*, 59, *60*, *77*
retrieve_tempdata, *30*, *55*, *57–59*, 60, *77*
retrieveall_appdata, *30*, *55*, *57, 58*, 58, *59,
60*, *77*
run_kindisperse, 61

sample_kindist, *32*, *40*, *43*, 61
sampledims (filter_methods), 32
sampledims,KinPairSimulation-method
(filter_methods), 32
sampledims<- (filter_methods), 32
sampledims<-,KinPairSimulation-method
(filter_methods), 32
samplenum (filter_methods), 32
samplenum,KinPairSimulation-method
(filter_methods), 32
samplenum<- (filter_methods), 32
samplenum<-,KinPairSimulation-method
(filter_methods), 32
sampling_stage, *25*, *28*, 63
sampling_stage,DispersalModel-method
(sampling_stage), 63
sampling_stage,KinPairData-method
(sampling_stage), 63
sampling_stage<- (sampling_stage), 63
sampling_stage<-,DispersalModel-method
(sampling_stage), 63
show,DispersalModel-method
(DispersalModel-class), 23
show,KinPairData-method
(KinPairData-class), 39
show,KinPairSimulation-method
(KinPairSimulation-class), 41
simdims, 64
simdims,KinPairSimulation-method
(simdims), 64
simdims<- (simdims), 64
simgraph_data, 65, *67*
simgraph_graph, *65*, 66
simtype, *5*, *34*, *38, 39*, 67
simtype,KinPairSimulation-method
(simtype), 67
simtype<- (simtype), 67
simulate_kindist_composite, *43*, *62*, 68,
73, *75*
simulate_kindist_custom, *23–26*, *28*, *43*,
70, 70, *75*
simulate_kindist_simple, *43*, *70*, *73*, 73
spacing (filter_methods), 32
spacing,KinPairSimulation-method
(filter_methods), 32
spacing<- (filter_methods), 32
spacing<-,KinPairSimulation-method
(filter_methods), 32
stages, 75
stages,DispersalModel-method (stages),
75

tsv_to_kinpair, *22*, *56*, 76, *78*

unmount_appdata, *30*, *55*, *57–60*, 77
upper (filter_methods), 32
upper,KinPairSimulation-method
(filter_methods), 32
upper<- (filter_methods), 32
upper<-,KinPairSimulation-method
(filter_methods), 32

vector_to_kinpair, *22*, *56*, *76*, 78
visible_stage, 78
visible_stage,DispersalModel-method
(visible_stage), 78

write_kindata, *49, 50*, 79