Package 'bnpsd'

October 12, 2022

Title Simulate Genotypes from the BN-PSD Admixture Model

Version 1.3.13

Description The Pritchard-Stephens-Donnelly (PSD) admixture model has k intermediate subpopulations from which n individuals draw their alleles dictated by their individual-specific admixture proportions. The BN-PSD model additionally imposes the Balding-Nichols (BN) allele frequency model to the intermediate populations, which therefore evolved independently from a common ancestral population T with subpopulation-specific FST (Wright's fixation index) parameters. The BN-PSD model can be used to yield complex population structures. This simulation approach is now extended to subpopulations related by a tree. Method described in Ochoa and Storey (2021) <doi:10.1371/journal.pgen.1009241>.

Depends

Imports stats, ape, nnls

Suggests popkin (>= 1.3.9), testthat, knitr, rmarkdown, RColorBrewer

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.1.1

VignetteBuilder knitr

URL https://github.com/StoreyLab/bnpsd/

BugReports https://github.com/StoreyLab/bnpsd/issues

NeedsCompilation no

Author Alejandro Ochoa [aut, cre] (<https://orcid.org/0000-0003-4928-3403>), John D. Storey [aut] (<https://orcid.org/0000-0001-5992-402X>)

Maintainer Alejandro Ochoa <alejandro.ochoa@duke.edu>

Repository CRAN

Date/Publication 2021-08-25 12:50:26 UTC

R topics documented:

`admix_prop_1d_circular` *Construct admixture proportion matrix for circular 1D geography*

Description

Assumes `k_subpops` intermediate subpopulations placed along a circumference (the `[0, 2 * pi]` line that wraps around) with even spacing spread by random walks (see details below), then `n_ind` individuals sampled equally spaced in `[coord_ind_first,coord_ind_last]` (default `[0, 2 * pi]` with a small gap so first and last individual do not overlap) draw their admixture proportions relative to the Von Mises density that models the random walks of each of these intermediate subpopulations. The spread of the random walks is `sigma = 1 / sqrt(kappa)` of the Von Mises density. If `sigma` is missing, it can be set indirectly by providing three variables: (1) the desired bias coefficient `bias_coeff`, (2) the coancestry matrix of the intermediate subpopulations `coanc_subpops` (up to a scalar factor), and (3) the final `fst` of the admixed individuals (see details below).

Usage

```
admix_prop_1d_circular(
  n_ind,
  k_subpops,
  sigma = NA,
```

```
  coord_ind_first = 2 * pi/(2 * n_ind),
  coord_ind_last = 2 * pi * (1 - 1/(2 * n_ind)),
  bias_coeff = NA,
  coanc_subpops = NULL,
  fst = NA
)
```

Arguments

n_ind	Number of individuals
k_subpops	Number of intermediate subpopulations
sigma	Spread of intermediate subpopulations (approximate standard deviation of Von Mises densities, see above) The edge cases sigma = 0 and sigma = Inf are handled appropriately!
coord_ind_first	Location of first individual
coord_ind_last	Location of last individual
	OPTIONS FOR BIAS COEFFICIENT VERSION
bias_coeff	If sigma is NA, this bias coefficient is required.
coanc_subpops	If sigma is NA, this intermediate subpops coancestry is required. It can be provided as a k_subpops-by-k_subpops matrix, a length-k_subpops population inbreeding vector (for independent subpopulations, where between-subpop coancestries are zero) or scalar (if population inbreeding values are all equal and coancestries are zero). This coanc_subpops can be in the wrong scale (it cancels out in calculations), which is returned corrected, to result in the desired fst (next).
fst	If sigma is NA, this FST of the admixed individuals is required.

Details

Assuming the full range of [0, 2 * pi] is considered, and the first and last individuals do not overlap, the gap between individuals is delta = 2 * pi / n. To not have any individuals on the edge, we place the first individual at delta / 2 and the last at 2 * pi - delta / 2. The location of subpopulation j is delta / 2 + (j - 1/2) / k * (2 * pi - delta), chosen to agree with the default correspondence between individuals and subpopulations of the linear 1D geography admixture scenario (admix_prop_1d_linear()).

If sigma is NA, its value is determined from the desired bias_coeff, coanc_subpops up to a scalar factor, and fst. Uniform weights for the final generalized FST are assumed. The scale of coanc_subpops is irrelevant because it cancels out in bias_coeff; after sigma is found, coanc_subpops is rescaled to give the desired final FST. However, the function stops if any rescaled coanc_subpops values are greater than 1, which are not allowed since they are IBD probabilities.

Value

If sigma was provided, returns the n_ind-by-k_subpops admixture proportion matrix (admix_proportions). If sigma is missing, returns a named list containing:

- admix_proportions: the n_ind-by-k_subpops admixture proportion matrix. If coanc_subpops had names, they are copied to the columns of this matrix.
- coanc_subpops: the input coanc_subpops rescaled.
- sigma: the fit value of the spread of intermediate subpopulations
- coanc_factor: multiplicative factor used to rescale coanc_subpops

Examples

```
# admixture matrix for 1000 individuals drawing alleles from 10 subpops
# simple version: spread of about 2 standard deviations along the circular 1D geography
# (just set sigma)
admix_proportions <- admix_prop_1d_circular(n_ind = 1000, k_subpops = 10, sigma = 2)

# advanced version: a similar model but with a bias coefficient of exactly 1/2
# (must provide bias_coeff, coanc_subpops, and fst in lieu of sigma)
k_subpops <- 10
# FST vector for intermediate independent subpops, up to a factor (will be rescaled below)
coanc_subpops <- 1 : k_subpops
obj <- admix_prop_1d_circular(
    n_ind = 1000,
    k_subpops = k_subpops,
    bias_coeff = 0.5,
    coanc_subpops = coanc_subpops,
    fst = 0.1 # desired final FST of admixed individuals
)

# in this case return value is a named list with three items:
admix_proportions <- obj$admix_proportions

# rescaled coancestry data (matrix or vector) for intermediate subpops
coanc_subpops <- obj$coanc_subpops

# and the sigma that gives the desired bias_coeff and final FST
sigma <- obj$sigma
```

---

admix_prop_1d_linear *Construct admixture proportion matrix for 1D geography*

Description

Assumes k_subpops intermediate subpopulations placed along a line at locations 1 : k_subpops spread by random walks, then n_ind individuals equally spaced in [coord_ind_first,coord_ind_last] draw their admixture proportions relative to the Normal density that models the random walks of each of these intermediate subpopulations. The spread of the random walks (the standard deviation of the Normal densities) is sigma. If sigma is missing, it can be set indirectly by providing three variables: (1) the desired bias coefficient bias_coeff, (2) the coancestry matrix of the intermediate subpopulations coanc_subpops (up to a scalar factor), and (3) the final fst of the admixed individuals (see details below).

Usage

```
admix_prop_1d_linear(
  n_ind,
  k_subpops,
  sigma = NA,
  coord_ind_first = 0.5,
  coord_ind_last = k_subpops + 0.5,
  bias_coeff = NA,
  coanc_subpops = NULL,
  fst = NA
)
```

Arguments

n_ind	Number of individuals.
k_subpops	Number of intermediate subpopulations.
sigma	Spread of intermediate subpopulations (standard deviation of normal densities). The edge cases sigma = 0 and sigma = Inf are handled appropriately!
coord_ind_first	Location of first individual (default 0.5).
coord_ind_last	Location of last individual (default k_subpops + 0.5).
	OPTIONS FOR BIAS COEFFICIENT VERSION
bias_coeff	If sigma is NA, this bias coefficient is required.
coanc_subpops	If sigma is NA, this intermediate subpops coancestry is required. It can be provided as a k_subpops-by-k_subpops matrix, a length-k_subpops population inbreeding vector (for independent subpopulations, where between-subpop coancestries are zero) or scalar (if population inbreeding values are all equal and coancestries are zero). This coanc_subpops can be in the wrong scale (it cancels out in calculations), which is returned corrected, to result in the desired fst (next).
fst	If sigma is NA, this FST of the admixed individuals is required.

Details

If sigma is NA, its value is determined from the desired bias_coeff, coanc_subpops up to a scalar factor, and fst. Uniform weights for the final generalized FST are assumed. The scale of coanc_subpops is irrelevant because it cancels out in bias_coeff; after sigma is found, coanc_subpops is rescaled to give the desired final FST. However, the function stops if any rescaled coanc_subpops values are greater than 1, which are not allowed since they are IBD probabilities.

Value

If sigma was provided, returns the n_ind-by-k_subpops admixture proportion matrix (admix_proportions). If sigma is missing, returns a named list containing:

- admix_proportions: the n_ind-by-k_subpops admixture proportion matrix. If coanc_subpops had names, they are copied to the columns of this matrix.

- coanc_subpops: the input coanc_subpops rescaled.
- sigma: the fit value of the spread of intermediate subpopulations
- coanc_factor: multiplicative factor used to rescale coanc_subpops

Examples

```
# admixture matrix for 1000 individuals drawing alleles from 10 subpops
# simple version: spread of 2 standard deviations along the 1D geography
# (just set sigma)
admix_proportions <- admix_prop_1d_linear(n_ind = 1000, k_subpops = 10, sigma = 2)

# as sigma approaches zero, admix_proportions approaches the independent subpopulations matrix
admix_prop_1d_linear(n_ind = 10, k_subpops = 2, sigma = 0)

# advanced version: a similar model but with a bias coefficient of exactly 1/2
# (must provide bias_coeff, coanc_subpops, and fst in lieu of sigma)
k_subpops <- 10
# FST vector for intermediate independent subpops, up to a factor (will be rescaled below)
coanc_subpops <- 1 : k_subpops
obj <- admix_prop_1d_linear(
    n_ind = 1000,
    k_subpops = k_subpops,
    bias_coeff = 0.5,
    coanc_subpops = coanc_subpops,
    fst = 0.1 # desired final FST of admixed individuals
)

# in this case return value is a named list with three items:
# admixture proportions
admix_proportions <- obj$admix_proportions

# rescaled coancestry data (matrix or vector) for intermediate subpops
coanc_subpops <- obj$coanc_subpops

# and the sigma that gives the desired bias_coeff and final FST
sigma <- obj$sigma
```

---

admix_prop_indep_subpops

Construct admixture proportion matrix for independent subpopulations

---

Description

This function constructs an admixture proportion matrix where every individuals is actually unadmixed (draws its full ancestry from a single intermediate subpopulation). The inputs are the vector of subpopulation labels labs for every individual (length n), and the length-k vector of unique subpopulations subpops in the desired order. If subpops is missing, the sorted unique subpopulations observed in labs is used. This function returns the admixture proportion matrix, for each individual 1 for the column corresponding to its subpopulation, 0 otherwise.

Usage

```
admix_prop_indep_subpops(labs, subpops = sort(unique(labs)))
```

Arguments

labs	Length-n vector of subpopulation labels
subpops	Optional length-k vector of unique subpopulations in desired order. Stops if subpops does not contain all unique labels in labs (no error if subpops contains additional labels).

Value

The n-by-k admixture proportion matrix. The unique subpopulation labels are given in the column names.

Examples

```
# vector of subpopulation memberships
labs <- c(1, 1, 1, 2, 2, 3, 1)
# admixture matrix with subpopulations (along columns) sorted
admix_proportions <- admix_prop_indep_subpops(labs)

# declare subpopulations in custom order
subpops <- c(3, 1, 2)
# columns will be reordered to match subpops as provided
admix_proportions <- admix_prop_indep_subpops(labs, subpops)

# declare subpopulations with unobserved labels
subpops <- 1:5
# note columns 4 and 5 will be false for all individuals
admix_proportions <- admix_prop_indep_subpops(labs, subpops)
```

---

bnpsd *A package for modeling and simulating an admixed population*

Description

The underlying model is called the BN-PSD admixture model, which combines the Balding-Nichols (BN) allele frequency model for the intermediate subpopulations with the Pritchard-Stephens-Donnelly (PSD) model of individual-specific admixture proportions. The BN-PSD model enables the simulation of complex population structures, ideal for illustrating challenges in kinship coefficient and FST estimation. Simulated loci are drawn independently (in linkage equilibrium).

Author(s)

Maintainer: Alejandro Ochoa <alejandro.ochoa@duke.edu> (ORCID)

Authors:

- John D. Storey <jstorey@princeton.edu> (ORCID)

See Also

Useful links:

- https://github.com/StoreyLab/bnpsd/
- Report bugs at https://github.com/StoreyLab/bnpsd/issues

Examples

```
# dimensions of data/model
# number of loci
m_loci <- 10
# number of individuals
n_ind <- 5
# number of intermediate subpops
k_subpops <- 2

# define population structure
# FST values for k = 2 subpopulations
inbr_subpops <- c( 0.1, 0.3 )
# admixture proportions from 1D geography
admix_proportions <- admix_prop_1d_linear( n_ind, k_subpops, sigma = 1 )
# also available:
# - admix_prop_1d_circular
# - admix_prop_indep_subpops

# get pop structure parameters of the admixed individuals
# the coancestry matrix
coancestry <- coanc_admix( admix_proportions, inbr_subpops )
# FST of admixed individuals
Fst <- fst_admix( admix_proportions, inbr_subpops )

# draw all random allele freqs and genotypes
out <- draw_all_admix( admix_proportions, inbr_subpops, m_loci )
# genotypes
X <- out$X
# ancestral allele frequencies (AFs)
p_anc <- out$p_anc

# OR... draw each vector or matrix separately
# provided for additional flexibility
# ancestral AFs
p_anc <- draw_p_anc( m_loci )
# independent subpops (intermediate) AFs
p_subpops <- draw_p_subpops( p_anc, inbr_subpops )
# individual-specific AFs
p_ind <- make_p_ind_admix( p_subpops, admix_proportions )
# genotypes
X <- draw_genotypes_admix( p_ind )

### Examples with a tree for intermediate subpopulations

# tree allows for correlated subpopulations
```

```
# (prev examples had independent subpopulations)

# best to start by specifying tree in Newick string format
tree_str <- '(S1:0.1,(S2:0.1,S3:0.1)N1:0.1)T;'
# and turn it into `phylo` object using the `ape` package
library(ape)
tree_subpops <- read.tree( text = tree_str )
# true coancestry matrix corresponding to this tree
coanc_subpops <- coanc_tree( tree_subpops )

# admixture proportions from 1D geography
# (constructed again but for k=3 tree)
k_subpops <- nrow( coanc_subpops )
admix_proportions <- admix_prop_1d_linear( n_ind, k_subpops, sigma = 0.5 )

# get pop structure parameters of the admixed individuals
# the coancestry matrix
coancestry <- coanc_admix( admix_proportions, coanc_subpops )
# FST of admixed individuals
Fst <- fst_admix( admix_proportions, coanc_subpops )

# draw all random allele freqs and genotypes, tree version
out <- draw_all_admix( admix_proportions, tree_subpops = tree_subpops, m_loci = m_loci )
# genotypes
X <- out$X
# ancestral allele frequencies (AFs)
p_anc <- out$p_anc

# OR... draw tree subpops (intermediate) AFs separately
p_subpops_tree <- draw_p_subpops_tree( p_anc, tree_subpops )
```

`coanc_admix` *Construct the coancestry matrix of an admixture model*

Description

The n-by-n coancestry matrix `Theta` of admixed individuals is determined by the n-by-k admixture proportion matrix `Q` and the k-by-k intermediate subpopulation coancestry matrix `Psi`, given by `Theta = Q %*% Psi %*% t(Q)`. In the more restricted BN-PSD model, `Psi` is a diagonal matrix (with FST values for the intermediate subpopulations along the diagonal, zero values off-diagonal).

Usage

```
coanc_admix(admix_proportions, coanc_subpops)
```

Arguments

`admix_proportions`
: The n-by-k admixture proportion matrix

coanc_subpops The intermediate subpopulation coancestry, given either as a k-by-k matrix (for the complete admixture model), or the length-k vector of intermediate subpopulation FST values (for the BN-PSD model; implies zero coancestry between subpopulations), or a scalar FST value shared by all intermediate subpopulations (also implies zero coancestry between subpopulations).

Details

As a precaution, function stops if both inputs have names and the column names of admix_proportions and the names in coanc_subpops disagree, which might be because these two matrices are not aligned or there is some other inconsistency.

Value

The n-by-n coancestry matrix.

Examples

```
# a trivial case: unadmixed individuals from independent subpopulations
# number of individuals and subpops
n_ind <- 5
# unadmixed individuals
admix_proportions <- diag(rep.int(1, n_ind))
# equal Fst for all subpops
coanc_subpops <- 0.2
# diagonal coancestry matryx
coancestry <- coanc_admix(admix_proportions, coanc_subpops)

# a more complicated admixture model
# number of individuals
n_ind <- 5
# number of intermediate subpops
k_subpops <- 2
# non-trivial admixture proportions
admix_proportions <- admix_prop_1d_linear(n_ind, k_subpops, sigma = 1)
# different Fst for each of the k_subpops
coanc_subpops <- c(0.1, 0.3)
# non-trivial coancestry matrix
coancestry <- coanc_admix(admix_proportions, coanc_subpops)
```

---

coanc_to_kinship *Transform coancestry matrix to kinship matrix*

Description

If Theta is the coancestry matrix and Phi is the kinship matrix (both are n-by-n symmetric), then these matrices agree off-diagonal, but the diagonal gets transformed as diag(Phi) = (1 + diag(Theta)) / 2.

Usage

```
coanc_to_kinship(coancestry)
```

Arguments

coancestry	The n-by-n coancestry matrix

Value

The n-by-n kinship matrix, preserving column and row names.

See Also

The inverse function is given by popkin::inbr_diag().

Examples

```
# a trivial case: unadmixed individuals from independent subpopulations
# number of individuals/subpops
n_ind <- 5
# unadmixed individuals
admix_proportions <- diag(rep.int(1, n_ind))
# equal Fst for all subpops
inbr_subpops <- 0.2
# diagonal coancestry matryx
coancestry <- coanc_admix(admix_proportions, inbr_subpops)
kinship <- coanc_to_kinship(coancestry)

# a more complicated admixture model
# number of individuals
n_ind <- 5
# number of intermediate subpops
k_subpops <- 2
# non-trivial admixture proportions
admix_proportions <- admix_prop_1d_linear(n_ind, k_subpops, sigma = 1)
# different Fst for each of the k subpops
inbr_subpops <- c(0.1, 0.3)
# non-trivial coancestry matrix
coancestry <- coanc_admix(admix_proportions, inbr_subpops)
kinship <- coanc_to_kinship( coancestry )
```

---

coanc_tree *Calculate coancestry matrix corresponding to a tree*

Description

This function calculates the coancestry matrix of the subpopulations which are tip nodes in the input tree. The edges of this tree are coancestry values relative to the parent node of each child node.

Usage

```
coanc_tree(tree)
```

Arguments

tree	The coancestry tree relating the k_subpops subpopulations. Must be a phylo object from the ape package (see ape::read.tree()). This tree may have a valid root edge (non-NULL tree$root.edge between 0 and 1), which is incorporated in the output calculations.

Details

The calculation takes into account that total coancestries are non-linear functions of the per-edge coancestries. Interestingly, the calculation can be simplified by a simple transformation performed by tree_additive(), see that for more information. The self-coancestry (diagonal values) are the total coancestries of the tip nodes. The coancestry between different subpopulations is the total coancestry of their last common ancestor node.

Value

The k_subpops-by-k_subpops coancestry matrix. The order of subpopulations along the rows and columns of this matrix matches tree$tip.label. The tip labels of the tree are copied to the row and column names of this matrix.

See Also

fit_tree() for the inverse function (when applied to coancestry matrices generated by a tree without noise).

tree_additive() for calculating the additive edges. This function is called internally by coanc_tree but the additive edges are not returned here, so call tree_additive() if you desired them.

Examples

```
# for simulating a tree with `rtree`
library(ape)

# a typical, non-trivial example
# number of tip subpopulations
k_subpops <- 3
# simulate a random tree
tree_subpops <- rtree( k_subpops )
# coancestry matrix of subpopulations
coancestry <- coanc_tree( tree_subpops )
```

`draw_all_admix` *Simulate random allele frequencies and genotypes from the BN-PSD admixture model*

Description

This function returns simulated ancestral, intermediate, and individual-specific allele frequencies and genotypes given the admixture structure, as determined by the admixture proportions and the vector or tree of intermediate subpopulation FST values. The function is a wrapper around `draw_p_anc()`, `draw_p_subpops()`/`draw_p_subpops_tree()`, `make_p_ind_admix()`, and `draw_genotypes_admix()` with additional features such as requiring polymorphic loci. Importantly, by default fixed loci (where all individuals were homozygous for the same allele) are re-drawn from the start (starting from the ancestral allele frequencies) so no fixed loci are in the output and no biases are introduced by re-drawing genotypes conditional on any of the previous allele frequencies (ancestral, intermediate, or individual-specific). Below `m_loci` (also `m`) is the number of loci, `n` is the number of individuals, and `k` is the number of intermediate subpopulations.

Usage

```
draw_all_admix(
  admix_proportions,
  inbr_subpops = NULL,
  m_loci,
  tree_subpops = NULL,
  want_genotypes = TRUE,
  want_p_ind = FALSE,
  want_p_subpops = FALSE,
  want_p_anc = TRUE,
  verbose = FALSE,
  require_polymorphic_loci = TRUE,
  maf_min = 0,
  beta = NA,
  p_anc = NULL,
  p_anc_distr = NULL
)
```

Arguments

`admix_proportions`	The `n`-by-`k` matrix of admixture proportions.
`inbr_subpops`	The length-`k` vector (or scalar) of intermediate subpopulation FST values. Either this or `tree_subpops` must be provided (but not both).
`m_loci`	The number of loci to draw.
`tree_subpops`	The coancestry tree relating the `k` intermediate subpopulations. Must be a `phylo` object from the `ape` package (see `ape::read.tree()`). Either this or `inbr_subpops` must be provided (but not both).

`want_genotypes` If TRUE (default), includes the matrix of random genotypes in the return list.

`want_p_ind` If TRUE (NOT default), includes the matrix of individual-specific allele frequencies in the return list. Note that by default `p_ind` is not constructed in full at all, instead a fast low-memory algorithm constructs it in parts as needed only; beware that setting `want_p_ind = TRUE` increases memory usage in comparison.

`want_p_subpops` If TRUE (NOT default), includes the matrix of random intermediate subpopulation allele frequencies in the return list.

`want_p_anc` If TRUE (default), includes the vector of random ancestral allele frequencies in the return list.

`verbose` If TRUE, prints messages for every stage in the algorithm.

`require_polymorphic_loci` If TRUE (default), returned genotype matrix will not include any fixed loci (loci that happened to be fixed are drawn again, starting from their ancestral allele frequencies, and checked iteratively until no fixed loci remain, so that the final number of polymorphic loci is exactly `m_loci`).

`maf_min` The minimum minor allele frequency (default zero), to extend the working definition of "fixed" above to include rare variants. This helps simulate a frequency-based locus ascertainment bias. Loci with minor allele frequencies less than or *equal* to this value are treated as fixed (passed to `fixed_loci()`). This parameter has no effect if `require_polymorphic_loci` is FALSE.

`beta` Shape parameter for a symmetric Beta for ancestral allele frequencies `p_anc`. If NA (default), `p_anc` is uniform with range in [0.01, 0.5]. Otherwise, `p_anc` has a symmetric Beta distribution with range in [0, 1]. Has no effect if either `p_anc` or `p_anc_distr` options are non-NULL.

`p_anc` If provided, it is used as the ancestral allele frequencies (instead of drawing random ones). Must either be a scalar or a length-`m_loci` vector. If scalar and `want_p_anc = TRUE`, then the returned `p_anc` is the scalar value repeated `m_loci` times (it is always a vector). If a locus was fixed and has to be redrawn, the ancestral allele frequency in `p_anc` is retained and only downstream allele frequencies and genotypes are redrawn (contrast to `p_anc_distr` below).

`p_anc_distr` If provided, ancestral allele frequencies are drawn with replacement from this vector (which may have any length) or function, instead of from `draw_p_anc()`. If a function, must accept a single parameter specifying the number of loci to draw. If a locus was fixed and has to be redrawn, the ancestral allele frequency is redrawn from the distribution (contrast to `p_anc` above).

Details

As a precaution, function stops if both the column names of `admix_proportions` and the names in `inbr_subpops` or `tree_subpops` exist and disagree, which might be because these two data are not aligned or there is some other inconsistency.

Value

A named list with the following items (which may be missing depending on options):

- `X`: An `m`-by-`n` matrix of genotypes. Included if `want_genotypes = TRUE`.

- p_anc: A length-m vector of ancestral allele frequencies. Included if want_p_anc = TRUE.
- p_subpops: An m-by-k matrix of intermediate subpopulation allele frequencies Included if want_p_subpops = TRUE.
- p_ind: An m-by-n matrix of individual-specific allele frequencies. Included if want_p_ind = TRUE.

Examples

```
# dimensions
# number of loci
m_loci <- 10
# number of individuals
n_ind <- 5
# number of intermediate subpops
k_subpops <- 2

# define population structure
# FST values for k = 2 subpopulations
inbr_subpops <- c(0.1, 0.3)
# admixture proportions from 1D geography
admix_proportions <- admix_prop_1d_linear(n_ind, k_subpops, sigma = 1)

# draw all random allele freqs and genotypes
out <- draw_all_admix(admix_proportions, inbr_subpops, m_loci)

# return value is a list with these items:

# genotypes
X <- out$X

# ancestral AFs
p_anc <- out$p_anc

# # these are excluded by default, but would be included if ...
# # ... `want_p_subpops == TRUE`
# # intermediate subpopulation AFs
# p_subpops <- out$p_subpops
#
# # ... `want_p_ind == TRUE`
# # individual-specific AFs
# p_ind <- out$p_ind
```

---

draw_genotypes_admix *Draw genotypes from the admixture model*

Description

Given the Individual-specific Allele Frequency (IAF) matrix `p_ind` for `m` loci (rows) and `n` individuals (columns), the genotype matrix `X` (same dimensions as `p_ind`) is drawn from the Binomial distribution equivalent to `X[ i, j ] <- rbinom( 1, 2, p_ind[ i, j ] )`, except the function is more efficient. If `admix_proportions` is provided as the second argument (a matrix with `n` individuals along rows and `k` intermediate subpopulations along the columns), the first argument `p_ind` is treated as the intermediate subpopulation allele frequency matrix (must be `m`-by-`k`) and the IAF matrix is equivalent to `p_ind %*% t( admix_proportions )`. However, in this case the function computes the IAF matrix in parts only, never stored in full, greatly reducing memory usage. If `admix_proportions` is missing, then `p_ind` is treated as the IAF matrix.

Usage

```
draw_genotypes_admix(p_ind, admix_proportions = NULL)
```

Arguments

`p_ind`	The `m`-by-`n` IAF matrix (if `admix_proportions` is missing) or the `m`-by-`k` intermediate subpopulation allele frequency matrix (if `admix_proportions` is present).
`admix_proportions`	The optional `n`-by-`k` admixture proportion matrix (to draw data from the admixture model using reduced memory, by not fully forming the IAF matrix). If provided, and if both `admix_proportions` and `p_ind` have column names, and if they disagree, the function stops as a precaution, as this suggests the data is misaligned or inconsistent in some way.

Value

The `m`-by-`n` genotype matrix. If `admix_proportions` is missing, the row and column names of `p_ind` are copied to this output. If `admix_proportions` is present, the row names of the output are the row names of `p_ind`, while the column names of the output are the row names of `admix_proportions`.

Examples

```
# dimensions
# number of loci
m_loci <- 10
# number of individuals
n_ind <- 5
# number of intermediate subpops
k_subpops <- 2

# define population structure
# FST values for k = 2 subpops
inbr_subpops <- c(0.1, 0.3)
# non-trivial admixture proportions
admix_proportions <- admix_prop_1d_linear(n_ind, k_subpops, sigma = 1)
```

```
# draw allele frequencies
# vector of ancestral allele frequencies
p_anc <- draw_p_anc(m_loci)

# matrix of intermediate subpop allele freqs
p_subpops <- draw_p_subpops(p_anc, inbr_subpops)

# matrix of individual-specific allele frequencies
p_ind <- make_p_ind_admix(p_subpops, admix_proportions)

# draw genotypes from intermediate subpops (one individual each)
X_subpops <- draw_genotypes_admix(p_subpops)

# and genotypes for admixed individuals
X_ind <- draw_genotypes_admix(p_ind)

# draw genotypes for admixed individuals without p_ind intermediate
# (p_ind is computed internally in parts, never stored in full,
# reducing memory use substantially)
X_ind <- draw_genotypes_admix(p_subpops, admix_proportions)
```

`draw_p_anc` *Draw random Uniform or Beta ancestral allele frequencies*

Description

This is simply a wrapper around `stats::runif()` or `stats::rbeta()` (depending on parameters) with different defaults and additional validations.

Usage

```
draw_p_anc(m_loci, p_min = 0.01, p_max = 0.5, beta = NA)
```

Arguments

`m_loci`	Number of loci to draw.
`p_min`	Minimum allele frequency to draw (Uniform case only).
`p_max`	Maximum allele frequency to draw (Uniform case only).
`beta`	Shape parameter for a symmetric Beta. If `NA` (default), Uniform(`p_min`, `p_max`) is used. Otherwise, a Symmetric Beta is used and the user-specified range is ignored (values in [0, 1] will be returned).

Value

A length-`m` vector of random ancestral allele frequencies

Examples

```
# Default is uniform with range between 0.01 and 0.5
p_anc <- draw_p_anc(m_loci = 10)

# Use of `beta` triggers a symmetric Beta distribution.
# This parameter has increased density for rare minor allele frequencies,
# resembling the 1000 Genomes allele frequency distribution
p_anc <- draw_p_anc(m_loci = 10, beta = 0.03)
```

---

draw_p_subpops *Draw allele frequencies for independent subpopulations*

Description

The allele frequency matrix `P` for `m_loci` loci (rows) and `k_subpops` independent subpopulations (columns) are drawn from the Balding-Nichols distribution with ancestral allele frequencies `p_anc` and FST parameters `inbr_subpops` equivalent to `P[ i, j ] <- rbeta( 1, nu_j * p_anc[i], nu_j * ( 1 - p_anc[i] ) )`, where `nu_j <- 1 / inbr_subpops[j] - 1`. The actual function is more efficient than the above code.

Usage

```
draw_p_subpops(p_anc, inbr_subpops, m_loci = NA, k_subpops = NA)
```

Arguments

`p_anc`	The scalar or length-`m_loci` vector of ancestral allele frequencies per locus.
`inbr_subpops`	The scalar or length-`k_subpops` vector of subpopulation FST values.
`m_loci`	If `p_anc` is scalar, optionally provide the desired number of loci (lest only one locus be simulated). Stops if both `length(p_anc) > 1` and `m_loci` is not `NA` and they disagree.
`k_subpops`	If `inbr_subpops` is a scalar, optionally provide the desired number of subpopulations (lest a single subpopulation be simulated). Stops if both `length(inbr_subpops) > 1` and `k_subpops` is not `NA` and they disagree.

Value

The `m_loci`-by-`k_subpops` matrix of independent subpopulation allele frequencies. If `p_anc` is length-`m_loci` with names, these are copied to the row names of this output matrix. If `inbr_subpops` is length-`k_subpops` with names, these are copied to the column names of this output matrix.

See Also

`draw_p_subpops_tree()` for version for subpopulations related by a tree, which can therefore be non-independent.

Examples

```
# a typical, non-trivial example
# number of loci
m_loci <- 10
# random vector of ancestral allele frequencies
p_anc <- draw_p_anc(m_loci)
# FST values for two subpops
inbr_subpops <- c(0.1, 0.3)
# matrix of intermediate subpop allele freqs
p_subpops <- draw_p_subpops(p_anc, inbr_subpops)

# special case of scalar p_anc
p_subpops <- draw_p_subpops(p_anc = 0.5, inbr_subpops, m_loci = m_loci)
stopifnot ( nrow( p_subpops ) == m_loci )

# special case of scalar inbr_subpops
k_subpops <- 2
p_subpops <- draw_p_subpops(p_anc, inbr_subpops = 0.2, k_subpops = k_subpops)
stopifnot ( ncol( p_subpops ) == k_subpops )

# both main parameters scalars but return value still matrix
p_subpops <- draw_p_subpops(p_anc = 0.5, inbr_subpops = 0.2, m_loci = m_loci, k_subpops = k_subpops)
stopifnot ( nrow( p_subpops ) == m_loci )
stopifnot ( ncol( p_subpops ) == k_subpops )

# passing scalar parameters without setting dimensions separately results in a 1x1 matrix
p_subpops <- draw_p_subpops(p_anc = 0.5, inbr_subpops = 0.2)
stopifnot ( nrow( p_subpops ) == 1 )
stopifnot ( ncol( p_subpops ) == 1 )
```

---

draw_p_subpops_tree *Draw allele frequencies for subpopulations related by a tree*

Description

The allele frequency matrix `P` for `m_loci` loci (rows) and `k_subpops` subpopulations (columns) are drawn from the Balding-Nichols distribution. If the allele frequency in the parent node is `p` and the FST parameter of the child node from the parent node is `F`, then the allele frequency in the child node is drawn from `rbeta( 1, nu * p, nu * ( 1 - p ) )`, where `nu <- 1 / F - 1`. This function iterates drawing allele frequencies through the tree structure, therefore allowing covariance between subpopulations that share branches.

Usage

```
draw_p_subpops_tree(p_anc, tree_subpops, m_loci = NA, nodes = FALSE)
```

Arguments

p_anc	The scalar or length-m_loci vector of ancestral allele frequencies per locus.
tree_subpops	The coancestry tree relating the k_subpops subpopulations. Must be a phylo object from the ape package (see ape::read.tree()). The edge lengths of this tree must be the FST parameters relating parent and child subpopulations.
m_loci	If p_anc is scalar, optionally provide the desired number of loci (lest only one locus be simulated). Stops if both length(p_anc) > 1 and m_loci is not NA and they disagree.
nodes	If FALSE (default), returns allele frequencies of "tip" subpopulations only; otherwise returns all allele frequencies, including internal nodes.

Value

The m_loci-by-k_subpops matrix of independent subpopulation allele frequencies. If nodes = FALSE, columns include only tip subpopulations. If nodes = TRUE, internal node subpopulations are also included (in this case the input p_anc is returned in the column corresponding to the root node). In all cases subpopulations are ordered as indexes in the tree, which normally implies the tip nodes are listed first, followed by the internal nodes (as in tree_subpops$edge matrix, see ape::read.tree() for details). The tree_subpops tip and node names are copied to the column names of this output matrix (individual names may be blank if missing in tree (such as for internal nodes); column names are NULL if all tree_subpops tip labels were blank, regardless of internal node labels). If p_anc is length-m_loci with names, these names are copied to the row names of this output matrix.

See Also

draw_p_subpops() for version for independent subpopulations.

For "phylo" tree class, see ape::read.tree()

Examples

```
# for simulating a tree with `rtree`
library(ape)

# a typical, non-trivial example
# number of tip subpopulations
k_subpops <- 3
# number of loci
m_loci <- 10
# random vector of ancestral allele frequencies
p_anc <- draw_p_anc(m_loci)
# simulate tree
tree_subpops <- rtree( k_subpops )
# matrix of intermediate subpop allele freqs
p_subpops <- draw_p_subpops_tree(p_anc, tree_subpops)
```

`fit_tree` *Fit a tree structure to a coancestry matrix*

Description

Implements a heuristic algorithm to find the optimal tree topology based on joining pairs of subpopulations with the highest between-coancestry values, and averaging parent coancestries for the merged nodes (a variant of WPGMA hierarchical clustering `stats::hclust()`). Branch lengths are optimized using the non-negative least squares approach `nnls::nnls()`, which minimize the residual square error between the given coancestry matrix and the model coancestry implied by the tree. This algorithm recovers the true tree when the input coancestry truly belongs to this tree and there is no estimation error (i.e. it is the inverse function of `coanc_tree()`), and performs well for coancestry estimates (i.e. the result of simulating genotypes from the true tree, i.e. from `draw_all_admix()`, and estimating the coancestry using `popkin::popkin()` followed by `popkin::inbr_diag()`).

Usage

```
fit_tree(coancestry, method = "mcquitty")
```

Arguments

`coancestry`	The coancestry matrix to fit a tree to.
`method`	The hierarchical clustering method (passed to `stats::hclust()`). While all `stats::hclust()` methods are supported, only two really make sense for this application: "mcquitty" (i.e. WPGMA, default) and "average" (UPGMA).

Details

The tree is bifurcating by construction, but edges may equal zero if needed, effectively resulting in multifurcation, although this code makes no attempt to merge nodes with zero edges. For best fit to arbitrary data, the root edge is always fit to the data (may also be zero). Data fit may be poor if the coancestry does not correspond to a tree, particularly if there is admixture between subpopulations.

Value

A `phylo` object from the `ape` package (see `ape::read.tree()`), with two additional list elements at the end:

- `edge`: (standard `phylo`.) A matrix describing the tree topology, listing parent and child node indexes.
- `Nnode`: (standard `phylo`.) Number of internal (non-leaf) nodes.
- `tip.label`: (standard `phylo`.) Labels for tips (leaf nodes), in order of index as in `edge` matrix above. These match the row names of the input `coancestry` matrix, or if names are missing, the row indexes of this matrix are used (in both cases, labels may be reordered compared to `rownames( coancestry )`). Tips are ordered as they appear in the above `edge` matrix (ensuring visual agreement in plots between the tree and its resulting coancestry matrix via

coanc_tree()), and in an order that matches the input coancestry's subpopulation order as much as possible (tree constraints do not permit arbitrary tip orders, but a heuristic implemented in tree_reorder() is used to determine a reasonable order when an exact match is not possible).

- edge.length: (standard phylo.) Values of edge lengths in same order as rows of edge matrix above.
- root.edge: (standard phylo.) Value of root edge length.
- rss: The residual sum of squares of the model coancestry versus the input coancestry.
- edge.length.add: Additive edge values (regarding their effect on coancestry, as opposed to edge.length which are probabilities, see coanc_tree())

See Also

coanc_tree()) for the inverse function (when the coancestry corresponds exactly to a tree).

tree_reorder() for reordering tree structure so that tips appear in a particular desired order.

Examples

```
# create a random tree
library(ape)
k <- 5
tree <- rtree( k )
# true coancestry matrix for this tree
coanc <- coanc_tree( tree )

# now recover tree from this coancestry matrix:
tree2 <- fit_tree( coanc )
# tree2 equals tree!
```

---

fixed_loci *Identify fixed loci*

Description

A locus is "fixed" if the non-missing sub-vector contains all 0's or all 2's (the locus is completely homozygous for one allele or completely homozygous for the other allele). This function tests each locus, returning a vector that is TRUE for each fixed locus, FALSE otherwise. Loci with only missing elements (NA) are treated as fixed. The parameter maf_min extends the "fixed" definition to loci whose minor allele frequency is smaller or equal than this value. Below m is the number of loci, and n is the number of individuals.

Usage

```
fixed_loci(X, maf_min = 0)
```

Arguments

X	The m-by-n genotype matrix
maf_min	The minimum minor allele frequency (default zero), to extend the working definition of "fixed" to include rare variants. Loci with minor allele frequencies less than or *equal* to this value are marked as fixed. Must be a scalar between 0 and 0.5.

Value

A length-m boolean vector where the i element is TRUE if locus i is fixed or completely missing, FALSE otherwise. If X had row names, they are copied to the names of this output vector.

Examples

```
# here's a toy genotype matrix
X <- matrix(
       data = c(
              2, 2, NA,  # fixed locus (with one missing element)
              0, NA, 0,  # another fixed locus, for opposite allele
              1, 1, 1,   # NOT fixed (heterozygotes are not considered fixed)
              0, 1, 2,   # a completely variable locus
              0, 0, 1,   # a somewhat "rare" variant
              NA, NA, NA # completely missing locus (will be treated as fixed)
             ),
       ncol = 3, byrow = TRUE)

# test that we get the desired values
stopifnot(
  fixed_loci(X) == c(TRUE, TRUE, FALSE, FALSE, FALSE, TRUE)
)

# the "rare" variant gets marked as "fixed" if we set `maf_min` to its frequency
stopifnot(
  fixed_loci(X, maf_min = 1/6) == c(TRUE, TRUE, FALSE, FALSE, TRUE, TRUE)
)
```

---

fst_admix *Calculate FST for the admixed individuals*

Description

This function returns the generalized FST of the admixed individuals given their admixture proportion matrix, the coancestry matrix of intermediate subpopulations (or its special cases, see coanc_subpops parameter below), and optional weights for individuals. This FST equals the weighted mean of the diagonal of the coancestry matrix (see coanc_admix()). Below there are n individuals and k intermediate subpopulations.

Usage

```
fst_admix(admix_proportions, coanc_subpops, weights = NULL)
```

Arguments

admix_proportions	The n-by-k admixture proportion matrix
coanc_subpops	Either the k-by-k intermediate subpopulation coancestry matrix (for the complete admixture model), or the length-k vector of intermediate subpopulation FST values (for the BN-PSD model; assumes zero coancestries between subpopulations), or a scalar FST value shared by all intermediate subpopulations (also assumes zero coancestry between subpopulations).
weights	Optional length-n vector of weights for individuals that define their generalized FST (default uniform weights)

Details

As a precaution, function stops if both inputs have names and the column names of admix_proportions and the names in coanc_subpops disagree, which might be because these two matrices are not aligned or there is some other inconsistency.

Value

The generalized FST of the admixed individuals

Examples

```
# set desired parameters
# number of individuals
n_ind <- 1000
# number of intermediate subpopulations
k_subpops <- 10

# differentiation of intermediate subpopulations
coanc_subpops <- ( 1 : k_subpops ) / k_subpops

# construct admixture proportions
admix_proportions <- admix_prop_1d_linear(n_ind, k_subpops, sigma = 1)

# lastly, calculate Fst!!! (uniform weights in this case)
fst_admix(admix_proportions, coanc_subpops)
```

make_p_ind_admix *Construct individual-specific allele frequency matrix under the PSD admixture model*

Description

Here m is the number of loci, n the number of individuals, and k the number of intermediate subpopulations. The m-by-n individual-specific allele frequency matrix p_ind is constructed from the m-by-k intermediate subpopulation allele frequency matrix p_subpops and the n-by-k admixture proportion matrix admix_proportions equivalent to p_ind <- p_subpops %*% t(admix_proportions). This function is a wrapper around tcrossprod(), but also ensures the output allele frequencies are in [0, 1] (otherwise not guaranteed due to limited machine precision).

Usage

```
make_p_ind_admix(p_subpops, admix_proportions)
```

Arguments

p_subpops	The m-by-k matrix of intermediate subpopulation allele frequencies.
admix_proportions	The n-by-k matrix of admixture proportions.

Details

If both admix_proportions and p_subpops have column names, and if they disagree, the function stops as a precaution, as this suggests the data is misaligned or inconsistent in some way.

Value

The m-by-n matrix of individual-specific allele frequencies p_ind. Row names equal those from p_subpops, and column names equal rownames from admix_proportions, if present.

Examples

```
# data dimensions
# number of loci
m_loci <- 10
# number of individuals
n_ind <- 5
# number of intermediate subpops
k_subpops <- 2

# FST values for k = 2 subpops
inbr_subpops <- c(0.1, 0.3)

# non-trivial admixture proportions
admix_proportions <- admix_prop_1d_linear(n_ind, k_subpops, sigma = 1)
```

```
# random vector of ancestral allele frequencies
p_anc <- draw_p_anc(m_loci)

# matrix of intermediate subpop allele freqs
p_subpops <- draw_p_subpops(p_anc, inbr_subpops)

# matrix of individual-specific allele frequencies
p_ind <- make_p_ind_admix(p_subpops, admix_proportions)
```

`scale_tree` *Scale a coancestry tree*

Description

Scale by a scalar `factor` all the edges (`$edge.length`) of a `phylo` object from the `ape` package, including the root edge (`$root.edge`) if present, and additive edges (`$edge.length.add`, present in trees returned by `fit_tree()`). Stops if any of the edges exceed 1 before or after scaling (since these edges are IBD probabilities).

Usage

```
scale_tree(tree, factor)
```

Arguments

`tree`	The coancestry tree to edit.
`factor`	The scalar factor to multiply all edges. Must be non-negative, and not be so large that any edge exceeds 1 after scaling.

Value

The edited tree with all edges scaled as desired.

See Also

`ape::read.tree()` for reading `phylo` objects and their definition.

Examples

```
# create a random tree
library(ape)
k <- 5
tree <- rtree( k )

# scale this tree
tree_scaled <- scale_tree( tree, 0.5 )
```

tree_additive *Calculate additive edges for a coancestry tree, or viceversa*

Description

A coancestry tree has IBD probabilities as edge values, which describe how each child and parent subpopulation in the tree is related. This means that each parameter is relative to its parent subpopulation (varies per edge), and they are not in general IBD probabilities from the root. This function computes "additive" edges that corresponds more closely with the coancestry matrix of this tree, which gives parameters relative to the root (ancestral) population (see details below). The additive edges are computed on a new element of the tree phylo object, so they do not overwrite the probabilistic edges. The reverse option assumes that the main edges of the phylo object are additive edges, then calculates the probabilistic edges and stores as the main edges and moves the original additive edges on the new element.

Usage

```
tree_additive(tree, rev = FALSE, force = FALSE)
```

Arguments

tree	The coancestry tree with either probabilistic edges (if rev = FALSE) or additive edges (if rev = TRUE) as the main edges (stored in tree$edge.length). Must be a phylo object from the ape package (see ape::read.tree()). This tree may have a valid root edge (non-NULL tree$root.edge between 0 and 1), which is incorporated in the output calculations. Function stops if the input data is not valid. Probabilistic edges are valid if and only if they are all between zero and one. Additive edges are valid if and only if they are all non-negative and the sum of edges between the root and each tip (leaf node) does not exceed 1.
rev	If FALSE (default), assumes the main edges are probabilistic values, and calculates additive values. If TRUE, assumes main edges are additive values, and calculates probabilistic values.
force	If FALSE (default), function stops if input tree already has additive edges (if tree$edge.length.add is not NULL). If TRUE, these values are ignored and overwritten.

Details

The calculation takes into account that total coancestries are non-linear combinations of the per-edge coancestries. For example, if the root node is A, and subpopulation C is connected to A only through an internal node B, then its total self-coancestry coanc_A_C relative to A is given by coanc_A_B (the coancestry between A and B) and coanc_B_C (the coancestry between B and C) by coanc_A_C = coanc_A_B + coanc_B_C * (1 - coanc_A_B). This transformation ensures that the total coancestry is a probability that does not exceed one even when the per-edge coancestries sum to a value greater than one. The "additive" edge for B and C is coanc_B_C * (1 - coanc_A_B), so it is the probabilistic edge coanc_B_C shrunk by 1 - coanc_A_B, which can then just be added to the parent edge coanc_A_B to give the total coancestry coanc_A_C. This transformation is iterated

for all nodes in the tree, noting that roots B connected to the root node A have equal probabilistic and additive edges coanc_A_B (unless the tree has a root edge, in which case that one is used to transform as above), and the edge of a node C not directly connected to a root uses the calculated edge coanc_A_C as above to shrink its children's edges into the additive scale.

Value

The input phylo object extended so that the main edges (tree$edge.length) are probabilistic edges, and the additive edges are stored in tree$edge.length.add. This is so for both values of rev

See Also

coanc_tree(), the key application facilitated by additive edges.

Examples

```
# for simulating a tree with `rtree`
library(ape)

# SETUP: number of tip subpopulations
k <- 5
# simulate a random tree
# edges are drawn from Uniform(0, 1), so these are valid probabilistic edges
tree <- rtree( k )
# inspect edges
tree$edge.length

# RUN calculate additive edges (safe to overwrite object)
tree <- tree_additive( tree )
# inspect edges again
# probabilistic edges are still main edges:
tree$edge.length
# additive edges are here
tree$edge.length.add


# let's go backwards now, starting from the additive edges
# SETUP
# these are harder to simulate, so let's copy the previous value to the main edges
tree$edge.length <- tree$edge.length.add
# and delete the extra entry (if it's present, function stops)
tree$edge.length.add <- NULL
# inspect before
tree$edge.length

# RUN reverse version (safe to overwrite object again)
tree <- tree_additive( tree, rev = TRUE )
# inspect after
# probabilistic edges are main edges:
tree$edge.length
# additive edges (previously main edges) were moved here
```

```
tree$edge.length.add
```

`tree_reindex_tips` *Reindex tree tips in order of appearance in edges*

Description

The `phylo` object from the `ape` package (see `ape::read.tree()`) permits mismatches between the order of tips as present in the tip labels vector (`tree$tip.label`) and in the edge matrix (`tree$edge`), which can cause mismatches in plots and other settings. This function reindexes edges and tips so that they agree in tip order.

Usage

```
tree_reindex_tips(tree)
```

Arguments

`tree`	A `phylo` object from the `ape` package (see `ape::read.tree()`). Works with standard `phylo` objects, and also with our extended trees (in that additive edges `tree$edge.length.add` are recalculated after reordering if they were present).

Details

Order mismatches between tip labels vector and edge matrix can lead to disagreeing order downstream, in variables that order tips as in the labels vector (such as the coancestry matrices output by our `coanc_tree()`) and tree plots (whose order is guided by the edge matrix, particularly when the edge matrix is ordered by clade as in `ape::reorder.phylo()`).

This function first reorders the edges of the input tree using `ape::reorder.phylo()` with default option `order = 'cladewise'`, which will list edges and tip nodes in plotting order. Then tips are indexed so that the first tip to appear has index 1 in the edge matrix (and consequently appears first in the tip labels vector), the second has index 2, and so on. Thus, the output tree has both edges and tips reordered, to have a consistent tip order and best experience visualizing trees and their coancestry matrices.

Value

The modified `tree` (`phylo` object) with reordered edges and tips.

See Also

`tree_reorder()` for reordering tree structure so that tips appear in a particular desired order.

Examples

```
# create a random tree
library(ape)
k <- 5
tree <- rtree( k )

# trees constructed this way are already ordered as desired, so this function doesn't change it:
tree2 <- tree_reindex_tips( tree )
# tree2 equals tree!

# let's scramble the edges on purpose
# to create an example where reindexing is needed

tree_rand <- tree
# new order of edges
indexes <- sample( Nedge( tree_rand ) )
# reorder all edge values
tree_rand$edge <- tree_rand$edge[ indexes, ]
tree_rand$edge.length <- tree_rand$edge.length[ indexes ]
# now let's reorder edges slightly so tree is more reasonable-looking
# (otherwise plot looks tangled)
tree_rand <- reorder( tree_rand, order = 'postorder' )
# you can now see that, unless permutation got lucky,
# the order of the tip labels in the vector and on the plot disagree:
tree_rand$tip.label
plot( tree_rand )

# now reorder tips to match plotting order (as defined in the `edge` matrix)
tree_rand <- tree_reindex_tips( tree_rand )
# now tip labels vector and plot should agree in order:
# (plot was not changed)
tree_rand$tip.label
plot( tree_rand )
```

---

tree_reorder *Reorder tree tips to best match a desired order*

Description

This functions reorganizes the tree structure so that its tips appear in a desired order if possible, or in a reasonably close order when an exact solution is impossible. This tip order in the output tree is the same in both the tip labels vector (`tree$tip.label`) and edge matrix (`tree$edge`), ensured by using `tree_reindex_tips()` internally.

Usage

```
tree_reorder(tree, labels)
```

Arguments

tree	A phylo object from the ape package (see ape::read.tree()). Works with standard phylo objects, and also with our extended trees (in that additive edges tree$edge.length.add are recalculated after reordering if they were present).
labels	A character vector with all tip labels in the desired order. Must contain each tip label in tree exactly once.

Details

This function has the same goal as ape::rotateConstr(), which implements a different heuristic algorithm that did not perform well in our experience.

Value

The modified tree (phylo object) with reordered edges and tips.

See Also

tree_reindex_tips() to reorder tips in the labels vector to match the edge matrix order, which ensures agreement in plots (assuming plot show desired order already).

Examples

```
# create a random tree
library(ape)
k <- 5
tree <- rtree( k )
# let's set the current labels as the desired order
labels <- tree$tip.label

# now let's scramble the edges on purpose
# to create an example where reordering is needed

tree_rand <- tree
# new order of edges
indexes <- sample( Nedge( tree_rand ) )
# reorder all edge values
tree_rand$edge <- tree_rand$edge[ indexes, ]
tree_rand$edge.length <- tree_rand$edge.length[ indexes ]
# now let's reorder edges slightly so tree is more reasonable-looking
# (otherwise plot looks tangled)
tree_rand <- reorder( tree_rand, order = 'postorder' )
# the order of the tip labels in the vector and on the plot disagree with each other:
tree_rand$tip.label
plot( tree_rand )

# now reorder tree object so tips are in the desired order:
tree_rand <- tree_reorder( tree_rand, labels )
# now tip labels vector and plot should agree in order:
# (the original tree was recovered!)
tree_rand$tip.label
```

```
plot( tree_rand )

# order the tree in a different way than the original order
labels <- paste0( 't', 1 : k )
# in this case, it's often impossible to get a perfect output order
# (because the tree structure constrains the possible plot orders),
# but this function tries its best to get close to the desired order
tree2 <- tree_reorder( tree, labels )
plot(tree2)
```

`undiff_af` *Undifferentiate an allele distribution*

Description

This function takes a vector of allele frequencies and a mean kinship value, and returns a new distribution of allele frequencies that is consistent with reversing differentiation with the given kinship, in the sense that the new distribution is more concentrated around the middle (0.5) than the input/original by an amount predicted from theory. The new distribution is created by weighing the input distribution with a random mixing distribution with a lower variance. An automatic method is provided that always selects a Beta distribution with just the right concentration to work for the given data and kinship. Explicit methods are also provided for more control, but are more likely to result in errors when mixing variances are not small enough (see details below).

Usage

```
undiff_af(
  p,
  kinship_mean,
  distr = c("auto", "uniform", "beta", "point"),
  alpha = 1,
  eps = 10 * .Machine$double.eps
)
```

Arguments

`p`	A vector of observed allele frequencies.
`kinship_mean`	The mean kinship value of the differentiation to reverse.
`distr`	Name of the mixing distribution to use. • "auto" picks a symmetric Beta distribution with parameters that ensure a small enough variance to succeed. • "beta" is a symmetric Beta distribution with parameter `alpha` as provided below. • "uniform" is a uniform distribution (same as "beta" with `alpha = 1`). • "point" is a distribution fully concentrated/fixed at 0.5 (same as the limit of "beta" with `alpha = Inf`, which has zero variance).

alpha	Shape parameter for distr = "beta", ignored otherwise.
eps	If distr = "auto", this small value is added to the calculated alpha to avoid roundoff errors and ensuring that the mixing variance is smaller than the maximum allowed.

Details

Model: Suppose we started from an allele frequency p0 with expectation 0.5 and variance V0. Differentiation creates a new (sample) allele frequency p1 without changing its mean (E(p1|p0) = p0) and with a conditional variance given by the mean kinship phi: Var(p1|p0) = p0*(1-p0)*phi. The total variance of the new distribution (calculated using the law of total variance) equals V1 = Var(p1) = phi/4 + (1-phi)*V0. (Also E(p1) = 0.5). So the new variance is greater for phi>0 (note V0 <= 1/4 for any distribution bounded in (0,1)). Thus, given V1 and phi, the goal is to construct a new distribution with the original, lower variance of V0 = (V1-phi/4)/(1-phi). An error is thrown if V1 < phi/4 in input data, which is inconsistent with this assumed model.

Construction of "undifferentiated" allele frequencies: p_out = w*p_in + (1-w)*p_mix, where p_in is the input with sample variance V_in (V1 in above model) and p_mix is a random draw from the mixing distribution distr with expectation 0.5 and known variance V_mix. The output variance is V_out = w^2*V_in + (1-w)^2*V_mix, which we set to the desired V_out = (V_in-phi/4)/(1-phi) (V0 in above model) and solve for w (the largest of the two quadratic roots is used). An error is thrown if V_mix > V_out (the output variance must be larger than the mixing variance). This error is avoided by manually adjusting choice of distr and alpha (for distr = "beta"), or preferably with distr = "auto" (default), which selects a Beta distribution with alpha = (1/(4*V_out)-1)/2 + eps that is guaranteed to work for any valid V_out (assuming V_in < phi/4).

Value

A list with the new distribution and several other informative statistics, which are named elements:

- p: A new vector of allele frequencies with the same length as input p, with the desired variance (see details) obtained by weighing the input p with new random data from distribution distr.
- w: The weight used for the input data (1-w for the mixing distribution).
- kinship_mean_max: The maximum mean kinship possible for undifferentiating this data (equals four times the input variance (see details), which results in zero output variance).
- V_in: sample variance of input p, assuming its expectation is 0.5.
- V_out: target variance of output p.
- V_mix: variance of mixing distribution.
- alpha: the value of alpha used for symmetric Beta mixing distribution, informative if distr = "auto".

Examples

```
# create random uniform data for these many loci
m <- 100
p <- runif( m )
# differentiate the distribution using Balding-Nichols model
F <- 0.1
```

```
nu <- 1 / F - 1
p2 <- rbeta( m, p * nu, (1 - p) * nu )

# now undifferentiate with this function
# NOTE in this particular case `F` is also the mean kinship
# default "automatic" distribution recommended
# (avoids possible errors for specific distributions)
p3 <- undiff_af( p2, F )$p

# note p3 does not equal p exactly (original is unrecoverable)
# but variances (assuming expectation is 0.5 for all) should be close to each other,
# and both be lower than p2's variance:
V1 <- mean( ( p - 0.5 )^2 )
V2 <- mean( ( p2 - 0.5 )^2 )
V3 <- mean( ( p3 - 0.5 )^2 )
# so p3 is stochastically consistent with p as far as the variance is concerned
```

Index