Package 'popkin'

December 18, 2019

Title Estimate Kinship and FST under Arbitrary Population Structure

Version 1.3.0

Description Provides functions to estimate the kinship matrix of individuals from a large set of bial-
lelic SNPs, and extract inbreeding coefficients and the generalized FST (Wright's fixation in-
dex). Method described in Ochoa and Storey (2016) <doi:10.1101/083923>.

Depends

Imports Rcpp (>= 0.12.10), RColorBrewer, graphics, grDevices

LinkingTo Rcpp, RcppEigen

Suggests BEDMatrix, testthat, knitr, rmarkdown, lfa

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

VignetteBuilder knitr

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

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

NeedsCompilation yes

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 2019-12-17 23:30:02 UTC

R topics documented:

---

popkin-package *A package for estimating kinship and FST under arbitrary population structure*

Description

The heart of this package is the popkin function, which estimates the kinship matrix of all individual pairs from their genotype matrix. Inbreeding coefficients, the generalized F_{ST}, and the individual-level pairwise F_{ST} matrix are extracted from the kinship matrix using inbr, fst, and pwfst, respectively. fst accepts weights for individuals to balance subpopulations obtained with weights_subpops. Kinship matrices can be renormalized (to change the most recent common ancestor population or MRCA) using rescale_popkin. Lastly, kinship and pairwise F_{ST} matrices can be visualized using plot_popkin (with the help of inbr_diag for kinship matrices only).

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

Examples

```
# estimate and visualize kinship and FST from a genotype matrix

# Construct toy data
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow = 3, byrow = TRUE) # genotype matrix
subpops <- c(1,1,2) # subpopulation assignments for individuals
subpops2 <- 1:3 # alternate labels treat every individual as a different subpop

# NOTE: for BED-formatted input, use BEDMatrix!
```

```
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
## X <- BEDMatrix(file) # load genotype matrix object

# estimate the kinship matrix from the genotypes "X"!
# all downstream analysis require "kinship", none use "X" after this
kinship <- popkin(X, subpops) # calculate kinship from X and optional subpop labels

# plot the kinship matrix, marking the subpopulations
# note inbr_diag replaces the diagonal of kinship with inbreeding coefficients
plot_popkin( inbr_diag(kinship), labs = subpops )

# extract inbreeding coefficients from kinship
inbreeding <- inbr(kinship)

# estimate FST
weights <- weights_subpops(subpops) # weigh individuals so subpopulations are balanced
Fst <- fst(kinship, weights) # use kinship matrix and weights to calculate fst
Fst <- fst(inbreeding, weights) # estimate more directly from inbreeding vector (same result)

# estimate and visualize the pairwise FST matrix
pairwise_fst <- pwfst(kinship) # estimated matrix
leg_title <- expression(paste('Pairwise ', F[ST])) # fancy legend label
# NOTE no need for inbr_diag() here!
plot_popkin(pairwise_fst, labs = subpops, leg_title = leg_title)

# rescale the kinship matrix using different subpopulations (implicitly changes the MRCA)
kinship2 <- rescale_popkin(kinship, subpops2)
```

`fst` *Calculate FST from a population-level kinship matrix or vector of inbreeding coefficients*

Description

This function simply returns the weighted mean inbreeding coefficient. If weights are `NULL` (default), the regular mean is calculated. If a kinship matrix is provided, then the inbreeding coefficients are extracted from its diagonal using `inbr` (requires the diagonal to contains self-kinship values ($\phi^T_{jj} = \frac{1}{2}(1 + f^T_j)$) as `popkin` returns, and not inbreeding coefficients (f^T_j) as `inbr_diag` returns).

Usage

```
fst(x, weights = NULL)
```

Arguments

`x`	The vector of inbreeding coefficients, or the kinship matrix if `x` is a matrix.
`weights`	Weights for individuals (optional, defaults to uniform weights)

Details

The returned weighted mean inbreeding coefficient equals the generalized F_{ST} if all individuals are "locally outbred" (i.e. if the self-relatedness of every individual stems entirely from the population structure rather than due partly to having unusually closely related parents, such as first or second cousins). Note most individuals in population-scale human data are locally outbred. If there are locally-inbred individuals, the returned value will overestimate F_{ST}.

Value

F_{ST}

Examples

```
# Get FST from a genotype matrix

# Construct toy data
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow = 3, byrow = TRUE) # genotype matrix
subpops <- c(1,1,2) # subpopulation assignments for individuals

# NOTE: for BED-formatted input, use BEDMatrix!
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
## X <- BEDMatrix(file) # load genotype matrix object

# estimate the kinship matrix "kinship" from the genotypes "X"!
kinship <- popkin(X, subpops) # calculate kinship from X and optional subpop labels
weights <- weights_subpops(subpops) # can weigh individuals so subpopulations are balanced
Fst <- fst(kinship, weights) # use kinship matrix and weights to calculate fst

Fst <- fst(kinship) # no (or NULL) weights implies uniform weights

inbr <- inbr(kinship) # if you extracted inbr for some other analysis...
Fst <- fst(inbr, weights) # ...use this inbreeding vector as input too!
```

---

inbr *Extract inbreeding coefficients from a kinship matrix*

Description

The kinship matrix contains inbreeding coefficients f_j^T along the diagonal, present as $\phi_{jj}^T = \frac{1}{2}(1 + f_j^T)$. This function extracts the vector of f_j^T values from the input kinship matrix.

Usage

```
inbr(kinship)
```

Arguments

kinship	The $n \times n$ kinship matrix.

Value

The length-n vector of inbreeding coefficient for each individual.

Examples

```
#########
# illustrate the main transformation on a 2x2 kinship matrix:
# same inbreeding values for both individuals
inbr <- 0.2
# corresponding self kinship (diagonal values) for both individuals
kinship_self <- (1 + inbr)/2
# actual kinship matrix (zero kinship between individuals)
kinship <- matrix(c(kinship_self, 0, 0, kinship_self), nrow=2)
# expected output of inbr (extracts inbreeding coefficients)
inbr_exp <- c(inbr, inbr)
# actual output from this function
inbr_obs <- inbr(kinship)
# verify that they match (up to machine precision)
stopifnot( all( abs(inbr_obs - inbr_exp) < .Machine$double.eps ) )

#########
# Construct toy data
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow=3, byrow=TRUE) # genotype matrix
subpops <- c(1,1,2) # subpopulation assignments for individuals

# NOTE: for BED-formatted input, use BEDMatrix!
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
## X <- BEDMatrix(file) # load genotype matrix object

# estimate the kinship matrix from the genotypes "X"!
kinship <- popkin(X, subpops) # calculate kinship from X and optional subpop labels

# extract inbreeding coefficients from Kinship
inbr <- inbr(kinship)
```

---

inbr_diag *Replace kinship diagonal with inbreeding coefficients*

Description

The usual kinship matrix contains self-kinship values $\phi_{jj}^T = \frac{1}{2}(1 + f_j^T)$ where f_j^T are inbreeding coefficients. This function returns a modified kinship matrix with each ϕ_{jj}^T replaced with f_j (off-diagonal $j \neq k$ values stay the same). The resulting matrix is better for visualization, but is not appropriate for modeling (e.g. in mixed-effects models for association or heritability estimation).

Usage

```
inbr_diag(kinship)
```

Arguments

kinship	A kinship matrix with self-kinship values along the diagonal. Can pass multiple kinship matrices contained in a list. If NULL, it is returned as-is.

Value

The modified kinship matrix, with inbreeding coefficients along the diagonal, preseving column and row names. If the input was a list of kinship matrices, the output is the corresponding list of transformed matrices. NULL inputs are preserved without causing errors.

See Also

The inverse function is given by coanc_to_kinship.

Examples

```
#########
# illustrate the main transformation on a 2x2 kinship matrix:
# same inbreeding values for both individuals
inbr <- 0.2
# corresponding self kinship (diagonal values) for both individuals
kinship_self <- (1 + inbr)/2
# kinship between the two individuals
kinship_between <- 0.1
# actual kinship matrix
kinship <- matrix(c(kinship_self, kinship_between, kinship_between, kinship_self), nrow=2)
# expected output of inbr_diag (replaces self kinship with inbreeding)
kinship_inbr_diag_exp <- matrix(c(inbr, kinship_between, kinship_between, inbr), nrow=2)
# actual output from this function
kinship_inbr_diag_obs <- inbr_diag(kinship)
# verify that they match (up to machine precision)
stopifnot( all( abs(kinship_inbr_diag_obs - kinship_inbr_diag_exp) < .Machine$double.eps ) )

# for a list of matrices, returns list of transformed matrices:
inbr_diag( list(kinship, kinship) )

# a list with NULL values also works
inbr_diag( list(kinship, NULL, kinship) )

#########
# Construct toy data (to more closely resemble real data analysis)
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow=3, byrow=TRUE) # genotype matrix
subpops <- c(1,1,2) # subpopulation assignments for individuals

# NOTE: for BED-formatted input, use BEDMatrix!
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
```

```
## X <- BEDMatrix(file) # load genotype matrix object

# estimate the kinship matrix from the genotypes "X"!
kinship <- popkin(X, subpops) # calculate kinship from X and optional subpop labels

# lastly, replace diagonal of kinship matrix with inbreeding coefficients
kinship_inbr_diag <- inbr_diag(kinship)
```

`mean_kinship` *Calculate the weighted mean kinship*

Description

This function computes a particular weighted mean kinship that arises in the context of kinship and F_{ST} estimators and in the definition of the effective sample size. This function allows for weights to be zero or even negative, but they are internally normalized to sum to one.

Usage

```
mean_kinship(kinship, weights = NULL)
```

Arguments

`kinship`	The kinship matrix
`weights`	Weights for individuals (optional). If `NULL` (default), uniform weights are used.

Value

The weighted mean kinship matrix, equivalent to `drop( weights %*% kinship %*% weights )` for normalized weights (which sum to one).

Examples

```
# construct a dummy kinship matrix
kinship <- matrix(c(0.5, 0, 0, 0.6), nrow=2)

# this is the ordinary mean
mean_kinship(kinship)

# weighted mean with twice as much weight on the second individual
# (weights are internally normalized to sum to one)
weights <- c(1, 2)
mean_kinship(kinship, weights)
```

n_eff *Calculates the effective sample size of the data*

Description

The effective sample size (n_{eff}) is the equivalent number of independent haplotypes that gives the same variance as that observed under the given population. The variance in question is for the weighted sample mean ancestral allele frequency estimator. It follows that n_{eff} equals the inverse of the weighted mean kinship. If `max=TRUE`, a calculation is performed that implicitly uses optimal weights which maximize n_{eff}: here n_{eff} equals the sum of the elements of the inverse kinship matrix. However, if `nonneg=TRUE` and if the above solution has negative weights (common), optimal non-negative weights are found instead (there are three algorithms available, see `algo`). If `max=FALSE`, then the input weights are used in this calculation, and if weights are `NULL`, uniform weights are used.

Usage

```
n_eff(
  kinship,
  max = TRUE,
  weights = NULL,
  nonneg = TRUE,
  algo = c("gradient", "newton", "heuristic"),
  tol = 1e-10
)
```

Arguments

`kinship`	An $n \times n$ kinship matrix.
`max`	If `TRUE`, returns the maximum n_{eff} value among those computed using all possible vectors of weights that sum to one (and which are additionally non-negative if `nonneg=TRUE`). If `FALSE`, n_{eff} is computed using the specific weight vector provided.
`weights`	Weights for individuals (optional). If `NULL`, uniform weights are used. This parameter is ignored if `max=TRUE`.
`nonneg`	If `TRUE` (default) and `max=TRUE`, non-negative weights that maximize n_{eff} are found. See `algo`. This has no effect if `max=FALSE`.
`algo`	Algorithm for finding optimal non-negative weights (applicable only if `nonneg=TRUE` and `max=TRUE` and the weights found by matrix inversion are non-negative). May be abbreviated. If "gradient" (default), an optimized gradient descent algorithm is used (fastest; recommended). If "newton", the exact multivariate newton's Method is used (slowest since $(n+1) \times (n+1)$ Hessian matrix needs to be inverted at every iteration; use if possible to confirm that "gradient" gives the best answer). If "heuristic", if the optimal solution by the inverse matrix method contains negative weights, the most negative weight in an iteration is forced to be zero in all subsequent iterations and the rest of the weights are solved for

using the inverse matrix method, repeating until all resulting weights are non-negative (also slow, since inversion of large matrices is required; least likely to find optimal solution).

`tol` Tolerance parameter for "gradient" and "newton" algorithms. The algorithms converge when the norm of the step vector is smaller than this tolerance value.

Details

The maximum n_{eff} possible is $2n$, where n is the number of individuals; this value is attained only when all haplotypes are independent (a completely unstructured population in Hardy-Weinberg equilibrium). The minimum n_{eff} possible is 1, which is attained in an extremely structured population with F_{ST} of 1, where every individual has exactly the same haplotype at every locus (no heterozygotes). Moreover, for K extremely-differentiated subpopulations (F_{ST}=1 per subpopulation) n_{eff} equals K. In this way, n_{eff} is smaller than the ideal value of $2n$ depending on the amount of kinship (covariance) in the data.

Occasionally, depending on the quality of the input kinship matrix, the estimated n_{eff} may be outside the theoretical [1, 2n] range, in which case the return value is set to the closest boundary value. The quality of the results depends on the success of matrix inversion (which for numerical reasons may incorrectly contain negative eigenvalues, for example) or of the gradient optimization.

Value

A list containing `n_eff` and `weights` (optimal weights if `max = TRUE`, input weights otherwise).

Examples

```
# Get n_eff from a genotype matrix

# Construct toy data
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow=3, byrow=TRUE) # genotype matrix
subpops <- c(1,1,2) # subpopulation assignments for individuals

# NOTE: for BED-formatted input, use BEDMatrix!
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
## X <- BEDMatrix(file) # load genotype matrix object

# estimate the kinship matrix "kinship" from the genotypes "X"!
kinship <- popkin(X, subpops) # calculate kinship from X and optional subpop labels
weights <- weights_subpops(subpops) # can weigh individuals so subpopulations are balanced

# use kinship matrix to calculate n_eff
# default mode returns maximum n_eff possible across all non-negative weights that sum to one
# also returns the weights that were optimal
obj <- n_eff(kinship)
n_eff_max <- obj$n_eff
w_max <- obj$weights

# version that uses weights provided
obj <- n_eff(kinship, max = FALSE, weights = weights)
n_eff_w <- obj$n_eff
```

```
w <- obj$weights # returns input weights renormalized for good measure

# no (or NULL) weights implies uniform weights
obj <- n_eff(kinship, max = FALSE)
n_eff_u <- obj$n_eff
w <- obj$weights # uniform weights
```

---

plot_popkin *Visualize one or more kinship matrices*

Description

This function plots one or more kinship matrices and a shared legend for the color key. Many options allow for fine control of individual or subpopulation labeling. This code assumes input matrices are symmetric.

Usage

```
plot_popkin(
  kinship,
  titles = NULL,
  col = NULL,
  col_n = 100,
  mar = NULL,
  mar_pad = 0.2,
  oma = 1.5,
  diag_line = FALSE,
  panel_letters = toupper(letters),
  panel_letters_cex = 1.5,
  ylab = "Individuals",
  ylab_adj = NA,
  ylab_line = 0,
  layout_add = TRUE,
  layout_rows = 1,
  leg_per_panel = FALSE,
  leg_title = "Kinship",
  leg_cex = 1,
  leg_n = 5,
  leg_mar = 3,
  leg_width = 0.3,
  names = FALSE,
  names_cex = 1,
  names_line = NA,
  names_las = 2,
  labs = NULL,
  labs_cex = 1,
```

```
  labs_las = 0,
  labs_line = 0,
  labs_sep = TRUE,
  labs_lwd = 1,
  labs_col = "black",
  labs_ticks = FALSE,
  labs_text = TRUE,
  labs_even = FALSE,
  null_panel_data = FALSE,
  weights = NULL,
  raster = is.null(weights),
  ...
)
```

Arguments

kinship	A numeric kinship matrix or a list of matrices. Note kinship may contain NULL elements (makes blank plots with titles; good for placeholders or non-existent data)
titles	Titles to add to each matrix panel (default is no titles)
col	Colors for heatmap (default is a red-white-blue palette symmetric about zero constructed using RColorBrewer).
col_n	The number of colors to use in the heatmap (applies if col = NULL).
mar	Margins shared by all panels (if a vector) or for each panel (if a list of such vectors). If the vector has length 1, mar corresponds to the shared lower and left margins, while the top and right margins are set to zero. If this length is 2, mar[1] is the same as above, while mar[2] is the top margin. If this length is 4, then mar is a fully-specified margin vector in the standard format c(bottom,left,top,right) that \link[graphics]{par}('mar') expects. Vectors of invalid lengths produce a warning. Note the padding mar_pad below is added to every margin if set. If NULL, the original margin values are used without change, and are reset for every panel that has a NULL value. The original margins are also reset after plotting is complete.
mar_pad	Margin padding added to all panels (mar above and leg_mar below). Default 0.2. Must be a scalar or a vector of length 4 to match \link[graphics]{par}('mar').
oma	Outer margin vector. If length 1, the value of oma is applied to the left outer margin only (so ylab below displays correctly) and zero outer margins elsewhere. If length 4, all outer margins are expected in standard format \link[graphics]{par}('mar') expects (see mar above). mar_pad above is never added to outer margins. If NULL, no outer margins are set (previous settings are preserved). Vectors of invalid lengths produce a warning.
diag_line	If TRUE adds a line along the diagonal (default no line). May also be a vector of logicals to set per panel (lengths must agree).
panel_letters	Vector of strings for labeling panels (default A-Z). No labels are added if NULL, or when there is only one panel except if its set to a single letter in that case (this behavior is useful if goal is to have multiple external panels but popkin only creates one of these panels).

panel_letters_cex
: Scaling factor of panel letters (default 1.5).

ylab
: The y-axis label (default "Individuals"). If `length(ylab) == 1`, the label is placed in the outer margin (shared across panels); otherwise `length(ylab)` must equal the number of panels and each label is placed in the inner margin of the respective panel.

ylab_adj
: The value of "adj" passed to \link[graphics]{mtext}. If `length(ylab) == 1`, only the first value is used, otherwise `length(ylab_adj)` must equal the number of panels.

ylab_line
: The value of "line" passed to \link[graphics]{mtext}. If `length(ylab) == 1`, only the first value is used, otherwise `length(ylab_line)` must equal the number of panels.

LAYOUT OPTIONS

layout_add
: If `TRUE` (default) then \link[graphics]{layout} is called internally with appropriate values for the required number of panels for each matrix, the desired number of rows (see `layout_rows` below) plus the color key legend. The original layout is reset when plotting is complete and if `layout_add = TRUE`. If a non-standard layout or additional panels (beyond those provided by `plot_popkin`) are desired, set to `FALSE` and call \link[graphics]{layout} yourself beforehand.

layout_rows
: Number of rows in layout, used only if `layout_add = TRUE`.

LEGEND (COLOR KEY) OPTIONS

leg_per_panel
: If `TRUE`, every kinship matrix get its own legend/color key (best for matrices with very different scales). If `FALSE` (default), a single legend/color key is shared by all kinship matrix panels.

leg_title
: The name of the variable that the heatmap colors measure (default "Kinship"), or a vector of such values if they vary per panel.

leg_cex
: Scaling factor for `leg_title` (default 1), or a vector of such values if they vary per panel.

leg_n
: The desired number of ticks in the legend y-axis (input to \link{pretty}, see that for more details), or a vector of such values if they vary per panel.

leg_mar
: Margin values for the legend panel only, or a list of such values if they vary per panel. A length-4 vector (in `c( bottom,left,top,right )` format that \link[graphics]{par}('mar') expects) specifies the full margins, to which `mar_pad` is added. Otherwise, the margins used in the last panel are preserved with the exception that the left margin is set to zero, and if `leg_mar` is length-1, it is used to specify the right margin (plus the value of `mar_pad`, see above).

INDIVIDUAL LABEL OPTIONS

leg_width
: The width of the legend panel, relative to the width of the kinship panel. This value is passed to \link[graphics]{layout} (ignored if `layout_add = FALSE`).

names
: If `TRUE`, the column and row names are plotted in the heatmap, or a vector of such values if they vary per panel.

names_cex
: Scaling factor for the column and row names, or a vector of such values if they vary per panel.

names_line
: Line where column and row names are placed, or a vector of such values if they vary per panel.

`names_las` Orientation of labels relative to axis. Default (2) makes labels perpendicular to axis.

SUBPOPULATION LABEL OPTIONS

`labs` Subpopulation labels for individuals. Use a matrix of labels to show groupings at more than one level (for a hierarchy or otherwise). If input is a vector or a matrix, the same subpopulation labels are shown for every heatmap panel; the input must be a list of such vectors or matrices if the labels vary per panel.

`labs_cex` A vector of label scaling factors for each level of labs, or a list of such vectors if labels vary per panel.

`labs_las` A vector of label orientations (in format that \link[graphics]{mtext} expects) for each level of labs, or a list of such vectors if labels vary per panel.

`labs_line` A vector of lines where labels are placed (in format that \link[graphics]{mtext} expects) for each level of labs, or a list of such vectors if labels vary per panel.

`labs_sep` A vector of logicals that specify whether lines separating the subpopulations are drawn for each level of labs, or a list of such vectors if labels vary per panel.

`labs_lwd` A vector of line widths for the lines that divide subpopulations (if `labs_sep = TRUE`) for each level of labs, or a list of such vectors if labels vary per panel.

`labs_col` A vector of colors for the lines that divide subpopulations (if `labs_sep = TRUE`) for each level of labs, or a list of such vectors if labels vary per panel.

`labs_ticks` A vector of logicals that specify whether ticks separating the subpopulations are drawn for each level of labs, or a list of such vectors if labels vary per panel.

`labs_text` A vector of logicals that specify whether the subpopulation labels are shown for each level of labs, or a list of such vectors if labels vary per panel. Useful for including separating lines or ticks without text.

`labs_even` A vector of logicals that specify whether the subpopulations labels are drawn with equal spacing for each level of labs, or a list of such vectors if labels vary per panel. When `TRUE`, lines mapping the equally-spaced labels to the unequally-spaced subsections of the heatmap are also drawn.

`null_panel_data`

If `FALSE` (default), panels with `NULL` kinship matrices must not have titles or other parameters set, and no panel letters are used in these cases. If `TRUE`, panels with `NULL` kinship matrices must have titles and other parameters set. In the latter case, these `NULL` panels also get panel letters. The difference is important when checking that lengths of non-singleton parameters agree.

`weights` A vector with weights for every individual, or a list of such vectors if they vary per panel. The width of every individual becomes proportional to their weight. Individuals with zero or negative weights are omitted.

`raster` A logical equivalent to `useRaster` option in the `image` function used internally, or a vector of such logicals if the choice varies per panel. If `weights` are non-`NULL` in a given panel, `raster = FALSE` is forced (this is necessary to plot images where columns and rows have variable width). If `weights` are `NULL`, the default is `raster = TRUE`, but in this case the user may override (for example, so panels are visually coherent when some use weights while others do not, as there are small differences in rendering implementation for each value of `raster`). Note

that a multipanel figure with a list of `weights` sets `raster = FALSE` to all panels by default, even if the weights were only applied to a subset of panels.

AXIS LABEL OPTIONS

`...` Additional options passed to \link[graphics]{image}. These are shared across panels

Details

`plot_popkin` plots the input kinship matrices as-is. For best results, a standard kinship matrix (such as the output of \link{popkin}) should have its diagonal rescaled to contain inbreeding coefficients (\link{inbr_diag} does this) before `plot_popkin` is used.

This function permits the labeling of individuals (from row and column names when `names = TRUE`) and of subpopulations (passed through `labs`). The difference is that the labels passed through `labs` are assumed to be shared by many individuals, and lines (or other optional visual aids) are added to demarcate these subgroups.

Examples

```
# Construct toy data
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow = 3, byrow = TRUE) # genotype matrix
subpops <- c(1,1,2) # subpopulation assignments for individuals

# NOTE: for BED-formatted input, use BEDMatrix!
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
## X <- BEDMatrix(file) # load genotype matrix object

# estimate the kinship matrix from the genotypes "X"!
kinship <- popkin(X, subpops) # calculate kinship from X and optional subpop labels

# simple plot of the kinship matrix, marking the subpopulations only
# note inbr_diag replaces the diagonal of kinship with inbreeding coefficients
# (see vignette for more elaborate examples)
plot_popkin( inbr_diag(kinship), labs = subpops )
```

`popkin` *Estimate kinship from a genotype matrix and subpopulation assignments*

Description

Given the biallelic genotypes of n individuals, this function returns the $n \times n$ kinship matrix such that the kinship estimate between the most distant subpopulations is zero on average (this sets the ancestral population T to the most recent common ancestor population).

Usage

```
popkin(
  X,
  subpops = NULL,
  n = NA,
  loci_on_cols = FALSE,
  mem_factor = 0.7,
  mem_lim = NA
)
```

Arguments

X	Genotype matrix, BEDMatrix object, or a function $X(m)$ that returns the genotypes of all individuals at m successive locus blocks each time it is called, and NULL when no loci are left.
subpops	The length-n vector of subpopulation assignments for each individual. If missing, every individual is effectively treated as a different population.
n	Number of individuals (required only when X is a function, ignored otherwise). If n is missing but subpops is not, n is taken to be the length of subpops.
loci_on_cols	If TRUE, X has loci on columns and individuals on rows; if false (the default), loci are on rows and individuals on columns. Has no effect if X is a function. If X is a BEDMatrix object, loci_on_cols = TRUE is set automatically.
mem_factor	Proportion of available memory to use loading and processing genotypes. Ignored if mem_lim is not NA.
mem_lim	Memory limit in GB, used to break up genotype data into chunks for very large datasets. Note memory usage is somewhat underestimated and is not controlled strictly. Default in Linux and Windows is mem_factor times the free system memory, otherwise it is 1GB (OSX and other systems).

Details

The subpopulation assignments are only used to estimate the baseline kinship (the zero value). If the user wants to re-estimate the kinship matrix using different subpopulation labels, it suffices to rescale it using rescale_popkin (as opposed to starting from the genotypes again, which gives the same answer less efficiently).

The matrix X must have values only in c(0,1,2,NA), encoded to count the number of reference alleles at the locus, or NA for missing data.

Value

The estimated $n \times n$ kinship matrix. If X has names for the individuals, they will be copied to the rows and columns of this kinship matrix.

Examples

```
# Construct toy data
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow = 3, byrow = TRUE) # genotype matrix
```

```
subpops <- c(1,1,2) # subpopulation assignments for individuals

# NOTE: for BED-formatted input, use BEDMatrix!
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
## X <- BEDMatrix(file) # load genotype matrix object

kinship <- popkin(X, subpops) # calculate kinship from genotypes and subpopulation labels
```

`pwfst` *Estimate the individual-level pairwise FST matrix*

Description

This function construct the individual-level pairwise F_{ST} matrix implied by the input kinship matrix. If the input is the true kinship matrix, the return value corresponds to the true pairwise F_{ST} matrix. On the other hand, if the input is the estimated kinship returned by `popkin`, then the return value is the pairwise F_{ST} estimates described in our paper. In all cases the diagonal of the pairwise F_{ST} matrix is zero by definition.

Usage

```
pwfst(kinship)
```

Arguments

`kinship`	The $n \times n$ kinship matrix

Value

The $n \times n$ pairwise F_{ST} matrix

Examples

```
# Construct toy data
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow=3, byrow=TRUE) # genotype matrix
subpops <- c(1,1,2) # subpopulation assignments for individuals

# NOTE: for BED-formatted input, use BEDMatrix!
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
## X <- BEDMatrix(file) # load genotype matrix object

# estimate the kinship matrix from the genotypes "X"!
kinship <- popkin(X, subpops) # calculate kinship from X and optional subpop labels

# lastly, compute pairwise FST matrix from the kinship matrix
pwF <- pwfst(kinship)
```

rescale_popkin *Rescale kinship matrix to set a given kinship value to zero.*

Description

Rescales the input kinship matrix Φ^T so that the value ϕ^T_{min} in the original kinship matrix becomes zero, using the formula

$$\Phi^{T'} = \frac{\Phi^T - \phi^T_{\text{min}}}{1 - \phi^T_{\text{min}}}.$$

This is equivalent to changing the ancestral population T into T' such that $\phi^{T'}_{\text{min}} = 0$. If subpopulation labels `subpops` are provided, they are used to estimate ϕ^T_{min}. If both `subpops` and `min_kinship` are provided, only `min_kinship` is used. If both `subpops` and `min_kinship` are omitted, the adjustment is equivalent to `min_kinship=min(kinship)`.

Usage

```
rescale_popkin(kinship, subpops = NULL, min_kinship = NA)
```

Arguments

`kinship`	An $n \times n$ kinship matrix.
`subpops`	The length-n vector of subpopulation assignments for each individual.
`min_kinship`	A scalar kinship value to define the new zero kinship.

Value

The rescaled $n \times n$ kinship matrix, with the desired level of relatedness set to zero.

Examples

```
# Construct toy data
X <- matrix(c(0,1,2,1,0,1,1,0,2), nrow=3, byrow=TRUE) # genotype matrix
subpops <- c(1,1,2) # subpopulation assignments for individuals
subpops2 <- 1:3 # alternate labels treat every individual as a different subpop

# NOTE: for BED-formatted input, use BEDMatrix!
# "file" is path to BED file (excluding .bed extension)
## library(BEDMatrix)
## X <- BEDMatrix(file) # load genotype matrix object

# suppose we first estimate kinship without subpopulations, which will be more biased
kinship <- popkin(X) # calculate kinship from genotypes, WITHOUT subpops
# then we visualize this matrix, figure out a reasonable subpopulation partition

# now we can adjust the kinship matrix!
kinship2 <- rescale_popkin(kinship, subpops)
# prev is faster but otherwise equivalent to re-estimating kinship from scratch with subpops:
```

```
# kinship2 <- popkin(X, subpops)

# can also manually set the level of relatedness min_kinship we want to be zero:
min_kinship <- min(kinship) # a naive choice for example
kinship2 <- rescale_popkin(kinship, min_kinship = min_kinship)

# lastly, omiting both subpops and min_kinship sets the minimum value in kinship to zero
kinship3 <- rescale_popkin(kinship2)
# equivalent to both of:
# kinship3 <- popkin(X)
# kinship3 <- rescale_popkin(kinship2, min_kinship = min(kinship))
```

`validate_kinship` *Validate a kinship matrix*

Description

Tests that the input is a valid kinship matrix (a numeric square R matrix). Throws errors if the input is not as above.

Usage

```
validate_kinship(kinship)
```

Arguments

`kinship`	The kinship matrix to validate.

Details

True kinship matrices have values strictly between 0 and 1, and diagonal values strictly between 0.5 and 1. However, estimated matrices may contain values slightly out of range. For greater flexibility, this function does not check for out-of-range values.

Value

Nothing

Examples

```
# this is a valid (positive) example
kinship <- matrix(c(0.5, 0, 0, 0.6), nrow=2)
# this will run without errors or warnings
validate_kinship(kinship)

# negative examples

# dies if input is missing
```

```
try( validate_kinship() )

# and if input is not a matrix
try( validate_kinship( 1:5 ) )

# and for non-numeric matrices
char_mat <- matrix(c('a', 'b', 'c', 'd'), nrow=2)
try( validate_kinship( char_mat ) )

# and non-square matrices
non_kinship <- matrix(1:2, nrow=2)
try( validate_kinship( non_kinship ) )
```

`weights_subpops` *Get weights for individuals that balance subpopulations*

Description

This function returns positive weights that sum to one for individuals using subpopulation labels, such that every subpopulation receives equal weight. In particular, if there are K subpopulations, then the sum of weights for every individuals of a given subpopulation will equal $\frac{1}{K}$. The weight of every individual is thus inversely proportional to the number of individuals in its subpopulation.

Usage

```
weights_subpops(subpops)
```

Arguments

subpops	The length-n vector of subpopulation assignments for each individual.

Value

The length-n vector of weights for each individual.

Examples

```
# if every individual has a different subpopulation, weights are uniform:
subpops <- 1:10
weights <- weights_subpops(subpops)
stopifnot(all(weights == rep.int(1/10,10)))

# subpopulations can be strings too
subpops <- c('a', 'b', 'c')
weights <- weights_subpops(subpops)
stopifnot(all(weights == rep.int(1/3,3)))

# if there are two subpopulations
# and the first has twice as many individuals as the second
```

```
# then the individuals in this first subpopulation weight half as much
# as the ones in the second subpopulation
subpops <- c(1, 1, 2)
weights <- weights_subpops(subpops)
stopifnot(all(weights == c(1/4,1/4,1/2)))
```

Index