Package 'popkin'

January 26, 2018

Title Estimate Kinship and FST under Arbitrary Population Structure

Version 1.0.5

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

LinkingTo Rcpp, RcppEigen

Suggests BEDMatrix, testthat, knitr, rmarkdown, lfa

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1.9000

VignetteBuilder knitr

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

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

NeedsCompilation yes

Author Alejandro Ochoa [aut, cre],
John D. Storey [aut]

Maintainer Alejandro Ochoa <ochoa@princeton.edu>

Repository CRAN

Date/Publication 2018-01-26 17:36:50 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 weightsSubpops. Kinship matrices can be renormalized (to change the most recent common ancestor population or MRCA) using rescalePopkin. Lastly, kinship and pairwise F_{ST} matrices can be visualized using plotPopkin (with the help of inbrDiag for kinship matrices only).

Author(s)

Maintainer: Alejandro Ochoa <ochoa@princeton.edu>

Authors:

- John D. Storey <jstorey@princeton.edu>

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

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

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

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

## estimate and visualize the pairwise FST matrix
pwF <- pwfst(Phi) # estimated matrix
legTitle <- expression(paste('Pairwise ', F[ST])) # fancy legend label
plotPopkin(pwF, labs=subpops, legTitle=legTitle) # NOTE no need for inbrDiag() here!

## rescale the kinship matrix using different subpopulations (implicitly changes the MRCA)
Phi2 <- rescalePopkin(Phi, subpops2)
```

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

Description

This function simply returns the weighted mean inbreeding coefficient f_j^T. If no weights are provided, the regular mean f_j^T is returned. If a kinship matrix Φ^T is provided, then f_j^T are extracted from its diagonal using `inbr` (assumes the diagonal of Φ^T is $\phi_{jj}^T = \frac{1}{2}(1 + f_j^T)$ as `popkin` returns, and not f_j^T as `inbrDiag` returns).

Usage

```
fst(x, w)
```

Arguments

`x`	The vector of inbreeding coefficients (f_j^T), or the kinship matrix Φ^T if `x` is a matrix.
`w`	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 "Phi" from the genotypes "X"!
Phi <- popkin(X, subpops) # calculate kinship from X and optional subpop labels
w <- weightsSubpops(subpops) # can weigh individuals so subpopulations are balanced
Fst <- fst(Phi, w) # use kinship matrix and weights to calculate fst

Fst <- fst(Phi) # no weights implies uniform weights

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

---

inbr *Extract inbreeding coefficients from a kinship matrix*

Description

The kinship matrix Φ^T 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 Φ^T.

Usage

```
inbr(Phi)
```

Arguments

Phi	The $n \times n$ kinship matrix Φ^T.

Value

The length-n vector of inbreeding coefficients f_j^T for each individual j.

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

## extract inbreeding coefficients from Phi
inbr <- inbr(Phi)
```

---

inbrDiag *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). This form produces more aesthetically pleasing visualizations, but is not appropriate for modeling (e.g. in GWAS or heritability estimation).

Usage

```
inbrDiag(Phi)
```

Arguments

Phi	The kinship matrix with self-kinship values along the diagonal

Value

The modified kinship matrix, with inbreeding coefficients along the diagonal

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

## lastly, replace diagonal of kinship matrix with inbreeding coefficients
PhiMod <- inbrDiag(Phi)
```

---

plotPopkin *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.

Usage

```
plotPopkin(x, titles = NULL, col = NULL, xMar = NULL, marPad = 0.2,
  diagLine = FALSE, panelLetters = toupper(letters), panelLetterCex = 1.5,
  ylab = "Individuals", ylabAdj = NA, ylabLine = 0, addLayout = TRUE,
  nr = 1, legTitle = "Kinship", legMar = NULL, nPretty = 5,
  showNames = FALSE, namesCex = 1, namesLine = NA, labs = NULL,
  labsCex = 1, labsLas = 0, labsLine = 0, labsSkipLines = FALSE,
  labsLwd = 1, labsCol = "black", labsDoTicks = FALSE,
  labsDoText = TRUE, labsEven = FALSE, ...)
```

Arguments

`x`	A numeric kinship matrix or a list of matrices
`titles`	Titles to add to each matrix panel (default is no titles)
`col`	Colors for heatmap
`xMar`	Margins for each panel (if a list) or for all panels (if a vector). Margins are in `c(bottom,left,top,right)` format that `par('mar')` expects. By default the existing margin values are used without change
`marPad`	Margin padding added to all panels (`xMar` above and `legMar` below). Default 0.2. Must be a scalar or a vector of length 4 to match `par('mar')`.

diagLine	If TRUE adds a line along the diagonal (default no line). May also be a vector of booleans to set per panel (lengths must agree)
panelLetters	Vector of strings for labeling panels (default A-Z). No labels are added when there is only one panel, or if 'panelLetters=NULL'
panelLetterCex	Scaling factor of panel letters (default 1.5)
	AXIS LABEL OPTIONS
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
ylabAdj	The value of "adj" passed to mtext. If length(ylab)==1, only the first value is used, otherwise length(ylabAdj) must equal the number of panels
ylabLine	The value of "line" passed to mtext. If length(ylab)==1, only the first value is used, otherwise length(ylabLine) must equal the number of panels
	LAYOUT OPTIONS
addLayout	If TRUE (default) then layout is called internally with appropriate values for the required number of panels for each matrix, the desired number of rows (see nr below) plus the color key legend. Set to FALSE and call layout beforehand if a non-standard layout or additional panels (beyond those provided by plotPopkin) are desired.
nr	Number of rows in layout, used only if addLayout=TRUE
	LEGEND (COLOR KEY) OPTIONS
legTitle	The name of the variable that the heatmap colors measure (default "Kinship")
legMar	Margin vector (in c(bottom,left,top,right) format that par('mar') expects) for the legend panel only. If not provided, the margins used in the last panel are preserved with the exception that the left margin is set to zero (plus the value of marPad, see above).
nPretty	The desired number of ticks in the legend y-axis (input to pretty, see that for more details)
	INDIVIDUAL LABEL OPTIONS
showNames	If TRUE, the column and row names are plotted in the heatmap
namesCex	Scaling factor for the column and row names
namesLine	Line where column and row names are placed
	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
labsCex	A vector of label scaling factors for each level of labs, or a list of such vectors if labels vary per panel
labsLas	A vector of label orientations (in format that mtext expects) for each level of labs, or a list of such vectors if labels vary per panel

labsLine	A vector of lines where labels are placed (in format that mtext expects) for each level of labs, or a list of such vectors if labels vary per panel
labsSkipLines	A vector of booleans 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
labsLwd	A vector of line widths for the lines that divide subpopulations (if labsSkipLines=FALSE) for each level of labs, or a list of such vectors if labels vary per panel
labsCol	A vector of colors for the lines that divide subpopulations (if labsSkipLines=FALSE) for each level of labs, or a list of such vectors if labels vary per panel
labsDoTicks	A vector of booleans 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
labsDoText	A vector of booleans 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.
labsEven	A vector of booleans 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
...	Additional options passed to image. These are shared across panels

Details

plotPopkin plots the input kinship matrices as-is. For best results, a standard kinship matrix (such as the output of popkin) should have its diagonal rescaled to contain inbreeding coefficients (inbrDiag does this) before plotPopkin is used.

This function permits the labeling of individuals (from row and column names when showNames=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.

For flexibility, this function will work for non-symmetric and even non-square matrices, even though proper kinship matrices are both. For non-symmetric inputs, differing rownames and colnames will display correctly (if showNames==TRUE). However, numerous options implicitly assume symmetry. For example, only the y-axis is labeled under the assumption that the x-axis is the same. Also, the same subpopulation labels are reproduced on both axes (for clarity).

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

```
## simple plot of the kinship matrix, marking the subpopulations only
## note inbrDiag replaces the diagonal of Phi with inbreeding coefficients
## (see vignette for more elaborate examples)
plotPopkin( inbrDiag(Phi), 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 Φ^T 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, lociOnCols = FALSE, memLim = 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.
lociOnCols	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, lociOnCols=TRUE is set automatically.
memLim	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 70 % of 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 Φ^T using different subpopulation labels, it suffices to rescale the given Φ^T using rescalePopkin (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 Φ^T.

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

Phi <- 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(Phi)
```

Arguments

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

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

`rescalePopkin` *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 `phiMin` are provided, only `phiMin` is used. If both `subpops` and `phiMin` are omitted, the adjustment is equivalent to `phiMin=min(Phi)`.

Usage

```
rescalePopkin(Phi, subpops = NULL, phiMin = NA)
```

Arguments

`Phi`	An $n \times n$ kinship matrix.
`subpops`	The length-n vector of subpopulation assignments for each individual.
`phiMin`	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
Phi <- 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!
Phi2 <- rescalePopkin(Phi, subpops)
## prev is faster but otherwise equivalent to re-estimating Phi from scratch with subpops:
## Phi2 <- popkin(X, subpops)

## can also manually set the level of relatedness phiMin we want to be zero:
phiMin <- min(Phi) # a naive choice for example
Phi2 <- rescalePopkin(Phi, phiMin=phiMin)

## lastly, omiting both subpops and phiMin sets the minimum value in Phi to zero
Phi3 <- rescalePopkin(Phi2)
## equivalent to both of:
## Phi3 <- popkin(X)
## Phi3 <- rescalePopkin(Phi2, phiMin=min(Phi))
```

`weightsSubpops` *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

```
weightsSubpops(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
w <- weightsSubpops(subpops)
stopifnot(all(w == rep.int(1/10,10)))
```

```
# subpopulations can be strings too
subpops <- c('a', 'b', 'c')
w <- weightsSubpops(subpops)
stopifnot(all(w == 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)
w <- weightsSubpops(subpops)
stopifnot(all(w == c(1/4,1/4,1/2)))
```

Index