Package 'GENESIS'

May 19, 2022

Type Package

Title GENetic EStimation and Inference in Structured samples (GENESIS): Statistical methods for analyzing genetic data from samples with population structure and/or relatedness

Version 2.26.0

Date 2021-04-19

Author Matthew P. Conomos, Stephanie M. Gogarten, Lisa Brown, Han Chen, Thomas Lumley, Kenneth Rice, Tamar Sofer, Adrienne Stilp, Timothy Thornton, Chaoyu Yu

Maintainer Stephanie M. Gogarten <sdmorris@uw.edu>

Description The GENESIS package provides methodology for estimating, inferring, and accounting for population and pedigree structure in genetic analyses. The current implementation provides functions to perform PC-AiR (Conomos et al., 2015, Gen Epi) and PC-Relate (Conomos et al., 2016, AJHG). PC-AiR performs a Principal Components Analysis on genome-wide SNP data for the detection of population structure in a sample that may contain known or cryptic relatedness. Unlike standard PCA, PC-AiR accounts for relatedness in the sample to provide accurate ancestry inference that is not confounded by family structure. PC-Relate uses ancestry representative principal components to adjust for population structure/ancestry and accurately estimate measures of recent genetic relatedness such as kinship coefficients, IBD sharing probabilities, and inbreeding coefficients. Additionally, functions are provided to perform efficient variance component estimation and mixed model association testing for both quantitative and binary phenotypes.

License GPL-3

URL https://github.com/UW-GAC/GENESIS

Depends

Imports Biobase, BiocGenerics, BiocParallel, GWASTools, gdsfmt, GenomicRanges, IRanges, S4Vectors, SeqArray, SeqVarTools, SNPRelate, data.table, graphics, grDevices, igraph, Matrix, methods, reshape2, stats, utils

Suggests CompQuadForm, COMPoissonReg, poibin, SPAtest, survey, testthat, BiocStyle, knitr, rmarkdown, GWASdata, dplyr, ggplot2, GGally, RColorBrewer, TxDb.Hsapiens.UCSC.hg19.knownGene

VignetteBuilder knitr

biocViews SNP, GeneticVariability, Genetics, StatisticalMethod, DimensionReduction, PrincipalComponent, GenomeWideAssociation, QualityControl, BiocViews

NeedsCompilation no

git_url https://git.bioconductor.org/packages/GENESIS

git_branch RELEASE_3_15

git_last_commit 1cffe35

git_last_commit_date 2022-04-26

Date/Publication 2022-05-19

R topics documented:

GENESIS-package *GENetic EStimation and Inference in Structured samples (GENESIS): Statistical methods for analyzing genetic data from samples with population structure and/or relatedness*

Description

The GENESIS package provides methodology for estimating, inferring, and accounting for population and pedigree structure in genetic analyses. The current implementation performs PC-AiR (Conomos et al., 2015, Gen Epi) and PC-Relate (Conomos et al., 2016, AJHG). PC-AiR performs a Principal Components Analysis on genome-wide SNP data for the detection of population structure in a sample that may contain known or cryptic relatedness. Unlike standard PCA, PC-AiR accounts for relatedness in the sample to provide accurate ancestry inference that is not confounded by family structure. PC-Relate uses ancestry representative principal components to adjust for population structure/ancestry and accurately estimate measures of recent genetic relatedness such as kinship coefficients, IBD sharing probabilities, and inbreeding coefficients. Additionally, functions are provided to perform efficient variance component estimation and mixed model association testing for both quantitative and binary phenotypes.

Details

The PC-AiR analysis is performed using the `pcair` function, which takes genotype data and pairwise measures of kinship and ancestry divergence as input and returns PC-AiR PCs as the ouput. The function `pcairPartition` is called within `pcair` and uses the PC-AiR algorithm to partition the sample into an ancestry representative 'unrelated subset' and 'related subset'. The function `plot.pcair` can be used to plot pairs of PCs from a class '`pcair`' object returned by the function `pcair`. The function `kingToMatrix` can be used to convert output text files from the KING software (Manichaikul et al., 2010) into an R matrix of pairwise kinship coefficient estimates in a format that can be used by the functions `pcair` and `pcairPartition`. The PC-Relate analysis is performed using the `pcrelate` function, which takes genotype data and PCs from PC-AiR and returns estimates of kinship coefficients, IBD sharing probabilities, and inbreeding coefficients. There are two functions required to perform SNP genotype association testing with mixed models. First, `fitNullModel` is called to fit the null model (i.e. no SNP genotype term) including fixed effects covariates, such as PC-AiR PCs, and random effects specified by their covariance structures, such as a kinship matrix created from PC-Relate output using `pcrelateToMatrix`. The function `fitNullModel` uses AIREML to estimate variance components for the random effects, and the function `varCompCI` can be used to find confidence intervals on the estimates as well as the proportion of total variability they explain; this allows for heritability estimation. Second, `assocTestSingle` is called with the null model output and the genotype data to perform either Wald or score based association tests.

Author(s)

Matthew P. Conomos, Stephanie M. Gogarten, Lisa Brown, Han Chen, Ken Rice, Tamar Sofer, Timothy Thornton, Chaoyu Yu

Maintainer: Stephanie M. Gogarten <sdmorris@uw.edu>

References

Conomos M.P., Miller M., & Thornton T. (2015). Robust Inference of Population Structure for Ancestry Prediction and Correction of Stratification in the Presence of Relatedness. Genetic Epidemiology, 39(4), 276-293.

Conomos M.P., Reiner A.P., Weir B.S., & Thornton T.A. (2016). Model-free Estimation of Recent Genetic Relatedness. American Journal of Human Genetics, 98(1), 127-148.

Manichaikul, A., Mychaleckyj, J.C., Rich, S.S., Daly, K., Sale, M., & Chen, W.M. (2010). Robust relationship inference in genome-wide association studies. Bioinformatics, 26(22), 2867-2873.

`admixMap` *admixMap*

Description

Run admixture analyses

Usage

```
admixMap(admixDataList, null.model, male.diploid=TRUE,
         genome.build=c("hg19", "hg38"),
         BPPARAM=bpparam(), verbose=TRUE)
```

Arguments

`admixDataList`	named list of `GenotypeIterator` or `SeqVarIterator` objects for each ancestry
`null.model`	A null model object returned by `fitNullModel`.
`male.diploid`	Logical for whether males on sex chromosomes are coded as diploid. Default is 'male.diploid=TRUE', meaning sex chromosome genotypes for males have values 0/2. If the input object codes males as 0/1 on sex chromosomes, set 'male.diploid=FALSE'.
`genome.build`	A character sting indicating genome build; used to identify pseudoautosomal regions on the X and Y chromosomes. These regions are not treated as sex chromosomes when calculating allele frequencies.
`BPPARAM`	A `BiocParallelParam` object to process blocks of variants in parallel. If not provided, the default back-end returned by `bpparam` will be used.
`verbose`	Logical indicator of whether updates from the function should be printed to the console; the default is TRUE.

Details

This function is used with local ancestry results such as those obtained from RFMix. RFMix output may be converted to PLINK format, and then to GDS with `snpgdsBED2GDS`.

admixDataList should have one value for each ancestry to be included in the test. The sum of all local ancestries at a particular locus must add up to 2, so if there are K ancestry groups, then only K-1 genotypes can be included since one local ancestry count can be written as a linear combination of all of the other local ancestry counts, resulting in collinearity and a matrix that won't be invertible.

See the example for how one might set up the admixDataList object. List names will propagate to the output file.

admixMap uses the BiocParallel package to process iterator chunks in parallel. See the BiocParallel documentation for more information on the default behaviour of bpparam and how to register different parallel backends. If serial execution is desired, set BPPARAM=BiocParallel::SerialParam(). Note that parallel execution requires more RAM than serial execution.

Value

A data.frame where each row refers to a different variant with the columns:

variant.id	The variant ID
chr	The chromosome value
pos	The base pair position
n.obs	The number of samples with non-missing genotypes
*.freq	The estimated frequency of alleles derived from each ancestry at that variant
*.Est	The effect size estimate for each additional copy of an allele derived from each ancestry, relative to the reference ancestry
*.SE	The estimated standard error of the effect size estimate for each ancestry
Joint.Stat	The chi-square Wald test statistic for the joint test of all local ancestry terms
Joint.pval	The Wald p-value for the joint test of all local ancestry terms

Author(s)

Matthew P. Conomos, Lisa Brown, Stephanie M. Gogarten, Tamar Sofer, Ken Rice, Chaoyu Yu

References

Brown, L.A. et al. (2017). Admixture Mapping Identifies an Amerindian Ancestry Locus Associated with Albuminuria in Hispanics in the United States. J Am Soc Nephrol. 28(7):2211-2220.

Maples, B.K. et al. (2013). RFMix: a discriminative modeling approach for rapid and robust local-ancestry inference. Am J Hum Genet. 93(2):278-88.

See Also

GenotypeIterator, fitNullModel, assocTestSingle

Examples

```
library(GWASTools)

# option 1: one GDS file per ancestry
afrfile <- system.file("extdata", "HapMap_ASW_MXL_local_afr.gds", package="GENESIS")
amerfile <- system.file("extdata", "HapMap_ASW_MXL_local_amer.gds", package="GENESIS")
eurfile <- system.file("extdata", "HapMap_ASW_MXL_local_eur.gds", package="GENESIS")
files <- list(afr=afrfile, amer=amerfile, eur=eurfile)
gdsList <- lapply(files, GdsGenotypeReader)

# make ScanAnnotationDataFrame
scanAnnot <- ScanAnnotationDataFrame(data.frame(
    scanID=getScanID(gdsList[[1]]), stringsAsFactors=FALSE))

# generate a phenotype
set.seed(4)
nsamp <- nrow(scanAnnot)
scanAnnot$pheno <- rnorm(nsamp, mean=0, sd=1)
set.seed(5)
scanAnnot$covar <- sample(0:1, nsamp, replace=TRUE)

genoDataList <- lapply(gdsList, GenotypeData, scanAnnot=scanAnnot)

# iterators
# if we have 3 ancestries total, only 2 should be included in test
genoIterators <- lapply(genoDataList[1:2], GenotypeBlockIterator)

# fit the null mixed model
null.model <- fitNullModel(scanAnnot, outcome="pheno", covars="covar")

# run the association test
myassoc <- admixMap(genoIterators, null.model,
                    BPPARAM=BiocParallel::SerialParam())
head(myassoc)

lapply(genoDataList, close)


# option 2: create a single file with multiple ancestries
# first, get dosages from all ancestries
library(gdsfmt)
dosages <- lapply(files, function(f) {
    gds <- openfn.gds(f)
    geno <- read.gdsn(index.gdsn(gds, "genotype"))
    closefn.gds(gds)
    geno
})
lapply(dosages, dim)

# create a new file with three dosage matrices, keeping all
# sample and snp nodes from one original file
tmpfile <- tempfile()
```

```
file.copy(afrfile, tmpfile)
gds <- openfn.gds(tmpfile, readonly=FALSE)
delete.gdsn(index.gdsn(gds, "genotype"))
add.gdsn(gds, "dosage_afr", dosages[["afr"]])
add.gdsn(gds, "dosage_amer", dosages[["amer"]])
add.gdsn(gds, "dosage_eur", dosages[["eur"]])
closefn.gds(gds)
cleanup.gds(tmpfile)

# read in GDS data, specifying the node for each ancestry
gds <- openfn.gds(tmpfile)
gds
genoDataList <- list()
for (anc in c("afr", "amer", "eur")){
  gdsr <- GdsGenotypeReader(gds, genotypeVar=paste0("dosage_", anc))
  genoDataList[[anc]] <- GenotypeData(gdsr, scanAnnot=scanAnnot)
}

# iterators
genoIterators <- lapply(genoDataList[1:2], GenotypeBlockIterator)

# run the association test
myassoc <- admixMap(genoIterators, null.model,
                    BPPARAM=BiocParallel::SerialParam())

close(genoDataList[[1]])
unlink(tmpfile)
```

---

assocTestAggregate *Aggregate Association Testing*

Description

assocTestAggregate performs aggregate association tests using the null model fit with fitNullModel.

Usage

```
## S4 method for signature 'SeqVarIterator'
assocTestAggregate(gdsobj, null.model, AF.max=1,
                   weight.beta=c(1,1), weight.user=NULL,
                   test=c("Burden", "SKAT", "fastSKAT", "SMMAT", "fastSMMAT",
                   "SKATO", "BinomiRare", "CMP"),
                   neig = 200, ntrace = 500,
                   rho = seq(from = 0, to = 1, by = 0.1),
                   sparse=TRUE, imputed=FALSE,
                   male.diploid=TRUE, genome.build=c("hg19", "hg38"),
                   BPPARAM=bpparam(), verbose=TRUE)
## S4 method for signature 'GenotypeIterator'
assocTestAggregate(gdsobj, null.model, AF.max=1,
```

```
    weight.beta=c(1,1), weight.user=NULL,
    test=c("Burden", "SKAT", "fastSKAT", "SMMAT", "fastSMMAT",
    "SKATO", "BinomiRare", "CMP"),
    neig = 200, ntrace = 500,
    rho = seq(from = 0, to = 1, by = 0.1),
    male.diploid=TRUE, BPPARAM=bpparam(), verbose=TRUE)
```

Arguments

gdsobj	An object of class SeqVarIterator from the package **SeqVarTools** containing the genotype data for the variants and samples to be used for the analysis.
null.model	A null model object returned by fitNullModel.
AF.max	A numeric value specifying the upper bound on the effect allele frequency for variants to be included in the analysis.
weight.beta	A numeric vector of length two specifying the two parameters of the Beta distribution used to determine variant weights; weights are given by dbeta(MAF, a, b), where MAF is the minor allele frequency, and a and b are the two parameters specified here. weight.beta = c(1,25) gives the Wu weights; weight.beta = c(0.5, 0.5) is proportional to the Madsen-Browning weights; and weight.beta = c(1,1) gives a weight of 1 to all variants. This input is ignored when weight.user is not NULL.
weight.user	A character string specifying the name of a variable to be used as variant weights. This variable can be in either 1) the variantData slot of gdsobj or 2) the mcols of the GRanges or GRangesList object used to create gdsobj (when gdsobj is a link{SeqVarRangeIterator} or link{SeqVarListIterator}). When left NULL (the default), the weights specified by weight.beta will be used.
test	A character string specifying the type of test to be performed. The possibilities are "Burden" (default), "SKAT", "fastSKAT", "SMMAT", "fastSMMAT", "BinomiRare", or "CMP".
neig	The number eigenvalues to approximate by using random projections for calculating p-values with fastSKAT or fastSMMAT; default is 200. See 'Details' for more information.
ntrace	The number of vectors to sample when using random projections to estimate the trace needed for p-value calculation with fastSKAT or fastSMMAT; default is 500. See 'Details' for more information.
rho	A numeric value (or vector of numeric values) in [0,1] specifying the rho parameter when using test == "SKATO"; these are the values for which SKAT-O is performed, defining the search space for the optimal rho. If rho = 0, this is equivalent to a standard SKAT test; if rho = 1, this is equivalent to a score burden test.
sparse	Logical indicator of whether to read genotypes as sparse Matrix objects; the default is TRUE. Set this to FALSE if the alternate allele dosage of the genotypes in the test are not expected to be mostly 0.
imputed	Logical indicator of whether to read dosages from the "DS" field containing imputed dosages instead of counting the number of alternate alleles.

male.diploid	Logical for whether males on sex chromosomes are coded as diploid. Default is 'male.diploid=TRUE', meaning sex chromosome genotypes for males have values 0/2. If the input gdsobj codes males as 0/1 on sex chromosomes, set 'male.diploid=FALSE'.
genome.build	A character sting indicating genome build; used to identify pseudoautosomal regions on the X and Y chromosomes. These regions are not treated as sex chromosomes when calculating allele frequencies.
BPPARAM	A BiocParallelParam object to process blocks of variants in parallel. If not provided, the default back-end returned by bpparam will be used.
verbose	Logical indicator of whether updates from the function should be printed to the console; the default is TRUE.

Details

The type of aggregate unit tested depends on the class of iterator used for gdsobj. Options include sliding windows, specific ranges of variants or selection of individual variants (ranges with width 1). See SeqVarIterator for more details.

assocTestAggregate uses the BiocParallel package to process iterator chunks in parallel. See the BiocParallel documentation for more information on the default behaviour of bpparam and how to register different parallel backends. If serial execution is desired, set BPPARAM=BiocParallel::SerialParam(). Note that parallel execution requires more RAM than serial execution.

All samples included in null model will be included in the association test, even if a different set of samples is present in the current filter for gdsobj.

The effect size estimate is for each copy of the alternate allele (when gdsobj is a SeqVarIterator object) or the "A" allele (when gdsobj is a GenotypeIterator object). We refer to this as the "effect allele" in the rest of the documentation. For multiallelic variants in SeqVarIterator objects, each alternate (or "A") allele is tested separately.

Monomorphic variants (including variants where every sample is a heterozygote) are always omitted from the aggregate unit prior to testing.

Somewhat similarly to SKAT-O, the variant Set Mixed Model Association Test (SMMAT, Chen et al., 2019) combines the burden test p-value with an adjusted SKAT (which is asymptotically independent of the burden test) p-value using a chi-square distribution with 4df from Fisher's method.

SKAT and SMMAT will attempt to use Davies' method (i.e. integration) to calculate p-values; if an error occurs in integration or the reported p-values are too small that they are unreliable (i.e. near machine epsilon), then the saddlepoint approximation will instead be used to calculate the p-values.

The fastSKAT method of Lumley et al. (2018) uses random matrix theory to speed up the computation of SKAT p-values. When test = "fastSKAT", the function attempts to inteligently determine which p-value calculation approach to use for each aggregation unit: (1) if min(number samples, number variants) is small enough, then the standard SKAT p-value calculation is used; (2) if min(number samples, number variants) is too large for standard SKAT, but small enough to explicitly compute the genotype covariance matrix, random projections are used to approximate the eigenvalues of the covariance matrix, and the fastSKAT p-value calculation is used; (3) if min(number samples, number variants) is too big to explicitly compute the genotype covariance matrix, random projections are used to approximate both the eigenvalues and the trace of the covariance matrix, and the fastSKAT p-value calculation is used.)

The fastSMMAT method uses the same random matrix theory as fastSKAT to speed up the computation of the p-value for the adjusted SKAT component of the test. When `test = "fastSMMAT"`, the function uses the same logic as for fastSKAT to determine which p-value calculation approach to use for each aggregation unit.

The BinomiRare test, run by using `test = "BinomiRare"`, and the CMP test, run by using `test = "CMP"` are carrier-only, robust tests. Only variants where the effect allele is minor will be tested. Both tests focuse on carriers of the rare variant allele ("carriers"), and use the estimated probabilities of the binary outcome within the carriers, estimated under the null of not association, and the actual number of observed outcomes, to compute p-values. BinomiRare uses the Poisson-Binomial distribution, and the CMP uses the Conway-Maxwell-Poisson distribution, and is specifically designed for mixed models. (If `test = "CMP"` but `null.model$family$mixedmodel = FALSE`, the BinomiRare test will be run instead.) These tests provide both a traditional p-value (`"pval"`) and a mid-p-value (`"midp"`), which is less conservative/more liberal, with the difference being more pronounced for small number of carriers. The BinomiRare test is described in Sofer (2017) and compared to the Score and SPA in various settings in Sofer and Guo (2020).

Value

A list with the following items:

`results` A data.frame containing the results from the main analysis. Each row is a separate aggregate test:

If `gdsobj` is a `SeqVarWindowIterator`:

`chr`	The chromosome value
`start`	The start position of the window
`end`	The end position of the window

Always:

`n.site`	The number of variant sites included in the test.
`n.alt`	The number of alternate (effect) alleles included in the test.
`n.sample.alt`	The number of samples with an observed alternate (effect) allele at any variant in the aggregate set.

If `test` is `"Burden"`:

`Score`	The value of the score function
`Score.SE`	The estimated standard error of the Score
`Score.Stat`	The score Z test statistic
`Score.pval`	The score p-value
`Est`	An approximation of the effect size estimate for each additional unit of burden
`Est.SE`	An approximation of the standard error of the effect size estimate
`PVE`	An approximation of the proportion of phenotype variance explained

If `test` is `"SKAT"` or `"fastSKAT"`:

`Q` The SKAT test statistic.

`pval` The SKAT p-value.

`err` Takes value 1 if there was an error in calculating the p-value; takes the value 2 when multiple random projections were required to get a good approximation from fastSKAT (the reported p-value is likely still reliable); 0 otherwise.

`pval.method` The p-value calculation method used. When standard SKAT is used, one of "integration" or "saddlepoint"; when fastSKAT random projections are used to approximate eigenvalues of the genotype covariance matrix, one of "ssvd_integration" or "ssvd_saddlepoint"; when fastSKAT random projections are used to approximate both the eigenvalues and the trace of the genotype covariance matrix, one of "rsvd_integration" or "rsvd_saddlepoint".

If `test` is `"SMMAT"` or `"fastSMMAT"`:

`pval_burden` The burden test p-value

`pval_theta` The p-value of the adjusted SKAT test (which is asymptotically independent of the burden test)

`pval_SMMAT` The SMMAT p-value after combining pval_burden and pval_theta using Fisher's method.

`err` Takes value 1 if there was an error calculating the SMMAT p-value; 0 otherwise. If `err=1`, `pval_SMMAT` is set to `pval_burden`.

`pval_theta.method`
The p-value calculation method used for `pval_theta` (the adjusted SKAT test). When standard SMMAT is used, one of "integration" or "saddlepoint"; when fastSMMAT random projections are used to approximate eigenvalues of the genotype covariance matrix, one of "ssvd_integration" or "ssvd_saddlepoint"; when fastSMMAT random projections are used to approximate both the eigenvalues and the trace of the genotype covariance matrix, one of "rsvd_integration" or "rsvd_saddlepoint".

If `test` is `"SKATO"`:

`Q_rho` The SKAT test statistic for the value of rho specified. There will be as many of these variables as there are rho values chosen.

`pval_rho` The SKAT p-value for the value of rho specified. There will be as many of these variables as there are rho values chosen.

`err_rho` Takes value 1 if there was an error in calculating the p-value for the value of rho specified when using the "kuonen" or "davies" methods; 0 otherwise. When there is an error, the p-value returned is from the "liu" method. There will be as many of these variables as there are rho values chosen.

`min.pval` The minimum p-value among the p-values calculated for each choice of rho.

`opt.rho` The optimal rho value; i.e. the rho value that gave the minimum p-value.

`pval_SKATO` The SKAT-O p-value after adjustment for searching across multiple rho values.

If `test` is `"BinomiRare"` or `"CMP"`:

`n.carrier` Number of individuals with at least one copy of the effect allele

n.D.carrier	Number of cases with at least one copy of the effect allele
pval	p-value
mid.pval	mid-p-value
variantInfo	A list with as many elements as aggregate tests performed. Each element of the list is a data.frame providing information on the variants used in the aggregate test with results presented in the corresponding row of results. Each of these data.frames has the following information:
variant.id	The variant ID
chr	The chromosome value
pos	The base pair position
allele.index	The index of the alternate allele. For biallelic variants, this will always be 1.
n.obs	The number of samples with non-missing genotypes
freq	The estimated effect allele frequency
MAC	The minor allele count. For multiallelic variants, "minor" is determined by comparing the count of the allele specified by allele.index with the sum of all other alleles.
weight	The weight assigned to the variant in the analysis.

Author(s)

Matthew P. Conomos, Stephanie M. Gogarten, Thomas Lumley, Tamar Sofer, Ken Rice, Chaoyu Yu, Han Chen

References

Leal, S.M. & Li, B. (2008). Methods for Detecting Associations with Rare Variants for Common Diseases: Application to Analysis of Sequence Data. American Journal of Human Genetics, 83(3): 311-321.

Browning, S.R. & Madsen, B.E. (2009). A Groupwise Association Test for Rare Mutations Using a Weighted Sum Statistic. PLoS Genetics, 5(2): e1000384.

Wu, M.C, Lee, S., Cai, T., Li, Y., Boehnke, M., & Lin, X. (2011). Rare-Variant Association Testing for Sequencing Data with the Sequence Kernel Association Test. American Journal of Human Genetics, 89(1): 82-93.

Lee, S. et al. (2012). Optimal Unified Approach for Rare-Variant Association Testing with Application to Small-Sample Case-Control Whole-Exome Sequencing Studies. American Journal of Human Genetics, 91(2): 224-237.

Chen, H., Huffman, J. E., Brody, J. A., Wang, C., Lee, S., Li, Z., ... & Blangero, J. (2019). Efficient variant set mixed model association tests for continuous and binary traits in large-scale whole-genome sequencing studies. The American Journal of Human Genetics, 104(2), 260-274.

Lumley, T., Brody, J., Peloso, G., Morrison, A., & Rice, K. (2018). FastSKAT: Sequence kernel association tests for very large sets of markers. Genetic epidemiology, 42(6), 516-527.

Examples

```
library(SeqVarTools)
library(Biobase)
library(GenomicRanges)

# open a sequencing GDS file
gdsfile <- seqExampleFileName("gds")
gds <- seqOpen(gdsfile)

# simulate some phenotype data
set.seed(4)
data(pedigree)
pedigree <- pedigree[match(seqGetData(gds, "sample.id"), pedigree$sample.id),]
pedigree$outcome <- rnorm(nrow(pedigree))

# construct a SeqVarData object
seqData <- SeqVarData(gds, sampleData=AnnotatedDataFrame(pedigree))

# fit the null model
nullmod <- fitNullModel(seqData, outcome="outcome", covars="sex")

# burden test - Range Iterator
gr <- GRanges(seqnames=rep(1,3), ranges=IRanges(start=c(1e6, 2e6, 3e6), width=1e6))
iterator <- SeqVarRangeIterator(seqData, variantRanges=gr)
assoc <- assocTestAggregate(iterator, nullmod, test="Burden",
                            BPPARAM=BiocParallel::SerialParam())
assoc$results
lapply(assoc$variantInfo, head)

# SKAT test - Window Iterator
seqSetFilterChrom(seqData, include="22")
iterator <- SeqVarWindowIterator(seqData)
assoc <- assocTestAggregate(iterator, nullmod, test="SKAT",
                            BPPARAM=BiocParallel::SerialParam())
head(assoc$results)
head(assoc$variantInfo)

# SKAT-O test - List Iterator
seqResetFilter(iterator)
gr <- GRangesList(
  GRanges(seqnames=rep(22,2), ranges=IRanges(start=c(16e6, 17e6), width=1e6)),
  GRanges(seqnames=rep(22,2), ranges=IRanges(start=c(18e6, 20e6), width=1e6)))
iterator <- SeqVarListIterator(seqData, variantRanges=gr)
assoc <- assocTestAggregate(iterator, nullmod, test="SKAT", rho=seq(0, 1, 0.25),
                            BPPARAM=BiocParallel::SerialParam())
assoc$results
assoc$variantInfo

# user-specified weights - option 1
seqResetFilter(iterator)
variant.id <- seqGetData(gds, "variant.id")
weights <- data.frame(variant.id, weight=runif(length(variant.id)))
```

```
variantData(seqData) <- AnnotatedDataFrame(weights)
iterator <- SeqVarListIterator(seqData, variantRanges=gr)
assoc <- assocTestAggregate(iterator, nullmod, test="Burden", weight.user="weight",
                            BPPARAM=BiocParallel::SerialParam())
assoc$results
assoc$variantInfo

# user-specified weights - option 2
seqResetFilter(iterator)
variantData(seqData)$weight <- NULL
gr <- GRangesList(
 GRanges(seqnames=rep(22,2), ranges=IRanges(start=c(16e6, 17e6), width=1e6), weight=runif(2)),
 GRanges(seqnames=rep(22,2), ranges=IRanges(start=c(18e6, 20e6), width=1e6), weight=runif(2)))
iterator <- SeqVarListIterator(seqData, variantRanges=gr)
assoc <- assocTestAggregate(iterator, nullmod, test="Burden", weight.user="weight",
                            BPPARAM=BiocParallel::SerialParam())
assoc$results
assoc$variantInfo

seqClose(seqData)
```

---

assocTestSingle *Genotype Association Testing with Mixed Models*

Description

assocTestSingle performs genotype association tests using the null model fit with fitNullModel.

Usage

```
## S4 method for signature 'SeqVarIterator'
assocTestSingle(gdsobj, null.model,
                test=c("Score", "Score.SPA", "BinomiRare", "CMP"),
                recalc.pval.thresh=0.05, fast.score.SE=FALSE,
                GxE=NULL,
                geno.coding=c("additive", "dominant", "recessive"),
                sparse=TRUE, imputed=FALSE,
                male.diploid=TRUE, genome.build=c("hg19", "hg38"),
                BPPARAM=bpparam(), verbose=TRUE)
## S4 method for signature 'GenotypeIterator'
assocTestSingle(gdsobj, null.model,
                test=c("Score", "Score.SPA", "BinomiRare", "CMP"),
                recalc.pval.thresh=0.05, GxE=NULL,
                geno.coding=c("additive", "dominant", "recessive"),
                male.diploid=TRUE, BPPARAM=bpparam(), verbose=TRUE)
```

Arguments

gdsobj	An object of class SeqVarIterator from the package **SeqVarTools**, or an object of class GenotypeIterator from the package **GWASTools**, containing the genotype data for the variants and samples to be used for the analysis.
null.model	A null model object returned by fitNullModel.
test	A character string specifying the type of test to be performed. The possibilities are "Score" (default), "Score.SPA", "BinomiRare", or "CMP"; "Score.SPA", "BinomiRare", and "CMP" can only be used when the family of the null model fit with fitNullModel is binomial.
recalc.pval.thresh	If test is not "Score", recalculate p-values using the specified 'test' for variants with a Score p-value below this threshold; return the score p-value for all other variants.
fast.score.SE	Logical indicator of whether to use the fast approximation of the score standard error for testing variant association. When FALSE (default), the true score SE is calculated. When TRUE, the fast score SE approximation from SAIGE is used. This option can only be used with a null model fit with fitNullModelFastScore or updated with nullModelFastScore. See 'Details' for further information.
GxE	A vector of character strings specifying the names of the variables for which a genotype interaction term should be included.If GxE is not NULL, test is ignored and Wald tests of interaction are performed. If GxE is NULL (default) no genotype interactions are included. See 'Details' for further information.
geno.coding	Whether genotypes should be coded as "additive" (0, 1, or 2 copies of the effect allele), "recessive" (1=homozygous for the effect allele, 0 otherwise), or "dominant" (1=heterozygous or homozygous for the effect allele, 0 for no effect allele). For recessive coding on sex chromosomes, males are coded as 1 if they are hemizygous for the effect allele.
sparse	Logical indicator of whether to read genotypes as sparse Matrix objects; the default is TRUE. Set this to FALSE if the alternate allele dosage of the genotypes in the test are not expected to be mostly 0.
imputed	Logical indicator of whether to read dosages from the "DS" field containing imputed dosages instead of counting the number of alternate alleles.
male.diploid	Logical for whether males on sex chromosomes are coded as diploid. Default is 'male.diploid=TRUE', meaning sex chromosome genotypes for males have values 0/2. If the input gdsobj codes males as 0/1 on sex chromosomes, set 'male.diploid=FALSE'.
genome.build	A character sting indicating genome build; used to identify pseudoautosomal regions on the X and Y chromosomes. These regions are not treated as sex chromosomes when calculating allele frequencies.
BPPARAM	A BiocParallelParam object to process blocks of variants in parallel. If not provided, the default back-end returned by bpparam will be used.
verbose	Logical indicator of whether updates from the function should be printed to the console; the default is TRUE.

Details

assocTestSingle uses the BiocParallel package to process iterator chunks in parallel. See the BiocParallel documentation for more information on the default behaviour of bpparam and how to register different parallel backends. If serial execution is desired, set BPPARAM=BiocParallel::SerialParam(). Note that parallel execution requires more RAM than serial execution.

All samples included in null model will be included in the association test, even if a different set of samples is present in the current filter for gdsobj.

The effect size estimate is for each copy of the alternate allele (when gdsobj is a SeqVarIterator object) or the "A" allele (when gdsobj is a GenotypeIterator object). We refer to this as the "effect allele" in the rest of the documentation. For multiallelic variants in SeqVarIterator objects, each alternate (or "A") allele is tested separately.

Sporadic missing genotype values are mean imputed using the allele frequency calculated on all other samples at that variant.

Monomorphic variants (including variants where every sample is a heterozygote) are omitted from the results.

The input GxE can be used to perform GxE tests. Multiple interaction variables may be specified, but all interaction variables specified must have been included as covariates in fitting the null model with fitNullModel. When performing GxE analyses, assocTestSingle will report two tests: (1) the joint Wald test of all genotype interaction terms in the model (this is the test for any genotype interaction effect), and (2) the joint Wald test of the genotype term along with all of the genotype interaction terms (this is the test for any genetic effect). Individual genotype interaction terms can be tested by creating test statistics from the reported effect size estimates and their standard errors (Note: when GxE contains a single continuous or binary covariate, this test is the same as the test for any genotype interaction effect mentioned above).

The saddle point approximation (SPA), run by using test = "Score.SPA", implements the method described by Dey et al. (2017), which was extended to mixed models by Zhou et al. (2018) in their SAIGE software. SPA provides better calibration of p-values when fitting mixed models with a binomial family for a sample with an imbalanced case to control ratio.

The fast approximation to the score standard error for testing variant association used by Zhou et al. (2018) in their SAIGE software is available by setting the fast.score.SE parameter to TRUE. This approximation may be much faster than computing the true score SE in large samples, as it replaces the full covariance matrix in the calculation with the product of a diagonal matrix and a scalar correction factor. This scalar correction factor must be computed beforehand and stored in the input null.model as se.correction, either by fitting the null.model with fitNullModelFastScore, or by updating a null.model previously fit with fitNullModel using the calcScore and nullModelFastScore functions. This approach assumes a constant scalar SE correction factor across all variants. This method is only available when gdsobj is a SeqVarIterator object.

The BinomiRare test, run by using test = "BinomiRare", and the CMP test, run by using test = "CMP" are carrier-only, robust tests. Only variants where the effect allele is minor will be tested. Both tests focuse on carriers of the rare variant allele ("carriers"), and use the estimated probabilities of the binary outcome within the carriers, estimated under the null of not association, and the actual number of observed outcomes, to compute p-values. BinomiRare uses the Poisson-Binomial distribution, and the CMP uses the Conway-Maxwell-Poisson distribution, and is specifically designed for mixed models. (If test = "CMP" but null.model$family$mixedmodel = FALSE, the

BinomiRare test will be run instead.) These tests provide both a traditional p-value ("pval") and a mid-p-value ("midp"), which is less conservative/more liberal, with the difference being more pronounced for small number of carriers. The BinomiRare test is described in Sofer (2017) and compared to the Score and SPA in various settings in Sofer and Guo (2020).

For the GenotypeIterator method, objects created with GdsGenotypeReader or MatrixGenotypeReader are supported. NcdfGenotypeReader objects are not supported.

Value

A data.frame where each row refers to a different variant with the columns:

variant.id	The variant ID
chr	The chromosome value
pos	The base pair position
allele.index	The index of the alternate allele. For biallelic variants, this will always be 1.
n.obs	The number of samples with non-missing genotypes
freq	The estimated effect allele frequency
MAC	The minor allele count. For multiallelic variants, "minor" is determined by comparing the count of the allele specified by allele.index with the sum of all other alleles.

If geno.coding is "recessive":

n.hom.eff	The number of samples homozygous for the effect allele.

If geno.coding is "dominant":

n.any.eff	The number of samples with any copies of the effect allele.

If test is "Score":

Score	The value of the score function
Score.SE	The estimated standard error of the Score
Score.Stat	The score Z test statistic
Score.pval	The score p-value
Est	An approximation of the effect size estimate for each additional copy of the effect allele
Est.SE	An approximation of the standard error of the effect size estimate
PVE	An approximation of the proportion of phenotype variance explained

If test is "Score.SPA":

SPA.pval	The score p-value after applying the saddle point approximation (SPA)
SPA.converged	logical indiactor of whether the SPA converged; NA indicates that the SPA was not applied and the original Score.pval was returned

If GxE is not NULL:

Est.G	The effect size estimate for the genotype term
Est.G:env	The effect size estimate for the genotype*env interaction term. There will be as many of these terms as there are interaction variables, and "env" will be replaced with the variable name.
SE.G	The estimated standard error of the genotype term effect size estimate
SE.G:env	The estimated standard error of the genotype*env effect size estimate. There will be as many of these terms as there are interaction variables, and "env" will be replaced with the variable name.
GxE.Stat	The Wald Z test statistic for the test of all genotype interaction terms. When there is only one genotype interaction term, this is the test statistic for that term.
GxE.pval	The Wald p-value for the test of all genotype interaction terms; i.e. the test of any genotype interaction effect
Joint.Stat	The Wald Z test statistic for the joint test of the genotype term and all of the genotype interaction terms
Joint.pval	The Wald p-value for the joint test of the genotype term and all of the genotype interaction terms; i.e. the test of any genotype effect

If `test` is `"BinomiRare"` or `"CMP"`:

n.carrier	Number of individuals with at least one copy of the effect allele
n.D.carrier	Number of cases with at least one copy of the effect allele
pval	p-value
mid.pval	mid-p-value

Author(s)

Matthew P. Conomos, Stephanie M. Gogarten, Tamar Sofer, Ken Rice, Chaoyu Yu

References

Dey, R., Schmidt, E. M., Abecasis, G. R., & Lee, S. (2017). A fast and accurate algorithm to test for binary phenotypes and its application to PheWAS. The American Journal of Human Genetics, 101(1), 37-49.

Sofer, T. (2017). BinomiRare: A robust test of the association of a rare variant with a disease for pooled analysis and meta-analysis, with application to the HCHS/SOL. Genetic Epidemiology, 41(5), 388-395.

Sofer, T. & Guo, N. (2020). Rare variants association testing for a binary outcome when pooling individual level data from heterogeneous studies. https://www.biorxiv.org/content/10.1101/2020.04.17.047530v1.

Zhou, W., Nielsen, J. B., Fritsche, L. G., Dey, R., Gabrielsen, M. E., Wolford, B. N., ... & Bastarache, L. A. (2018). Efficiently controlling for case-control imbalance and sample relatedness in large-scale genetic association studies. Nature genetics, 50(9), 1335.

See Also

`fitNullModel` for fitting the null mixed model needed as input to `assocTestSingle`. `SeqVarIterator` for creating the input object with genotypes. `effectAllele` for returning the effect allele for each variant.

Examples

```
library(SeqVarTools)
library(Biobase)

# open a sequencing GDS file
gdsfile <- seqExampleFileName("gds")
gds <- seqOpen(gdsfile)

# simulate some phenotype data
set.seed(4)
data(pedigree)
pedigree <- pedigree[match(seqGetData(gds, "sample.id"), pedigree$sample.id),]
pedigree$outcome <- rnorm(nrow(pedigree))

# construct a SeqVarIterator object
seqData <- SeqVarData(gds, sampleData=AnnotatedDataFrame(pedigree))
iterator <- SeqVarBlockIterator(seqData)

# fit the null model
nullmod <- fitNullModel(iterator, outcome="outcome", covars="sex")

# run the association test
assoc <- assocTestSingle(iterator, nullmod,
                         BPPARAM=BiocParallel::SerialParam())

# use fast score SE for a null model with a covariance matrix
seqResetFilter(seqData)
grm <- SNPRelate::snpgdsGRM(seqData, verbose=FALSE)
covmat <- grm$grm; dimnames(covmat) <- list(grm$sample.id, grm$sample.id)
set.seed(5)
nullmod <- fitNullModelFastScore(iterator, outcome="outcome", covars="sex", cov.mat=covmat)
assoc.se <- assocTestSingle(iterator, nullmod, fast.score.SE=TRUE,
                            BPPARAM=BiocParallel::SerialParam())

seqClose(iterator)


library(GWASTools)

# open a SNP-based GDS file
gdsfile <- system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS")
gds <- GdsGenotypeReader(filename = gdsfile)

# simulate some phenotype data
set.seed(4)
pheno <- data.frame(scanID=getScanID(gds),
                    outcome=rnorm(nscan(gds)))

# construct a GenotypeIterator object
genoData <- GenotypeData(gds, scanAnnot=ScanAnnotationDataFrame(pheno))
iterator <- GenotypeBlockIterator(genoData)
```

```
# fit the null model
nullmod <- fitNullModel(iterator, outcome="outcome")

# run the association test
assoc <- assocTestSingle(iterator, nullmod,
                         BPPARAM=BiocParallel::SerialParam())

close(iterator)
```

---

computeVSIF *Computes variant-specific inflation factors*

Description

Computes variant-specific inflation factors resulting from differences in variances and allele frequencies across groups pooled together in analysis.

Usage

```
computeVSIF(freq, n, sigma.sq)
computeVSIFNullModel(null.model, freq, group.var.vec)
```

Arguments

freq	A named vector or a matrix of effect allele frequencies across groups. Vector/column names are group names; rows (for a matrix) are variants.
n	A named vector of group sample sizes.
sigma.sq	A named vector of residual variances across groups.
null.model	A null model constructed with fitNullModel.
group.var.vec	A named vector of group membership. Names are sample.ids, values are group names.

Details

computeVSIF computes the expected inflation/deflation for each specific variant caused by differences in allele frequencies in combination with differences in residual variances across groups that are aggregated together (e.g. individuals with different genetic ancestry patterns). The inflation/deflation is especially expected if a homogeneous variance model is used.

computeVSIFNullModel uses the null model and vector of group membership to extract sample sizes and residual variances for each group. It then calls function computeVSIF to compute the inflation factors. The null model should be fit under a homogeneous variance model.

Value

`SE_true` Large sample test statistic variances accounting for differences in residual variances.

`SE_naive` Large sample test statistic variances (wrongly) assuming that all residual variances are the same across groups.

`Inflation_factor` Variant-specific inflation factors. Values higher than 1 suggest inflation (too significant p-value), values lower than 1 suggest deflation (too high p-value).

Author(s)

Tamar Sofer, Kenneth Rice

References

Sofer, T., Zheng, X., Laurie, C. A., Gogarten, S. M., Brody, J. A., Conomos, M. P., ... & Rice, K. M. (2020). Population Stratification at the Phenotypic Variance level and Implication for the Analysis of Whole Genome Sequencing Data from Multiple Studies. BioRxiv.

Examples

```
n <- c(2000, 5000, 100)
sigma.sq <- c(1, 1, 2)
freq.vec <- c(0.1, 0.2, 0.5)
names(freq.vec) <- names(n) <- names(sigma.sq) <- c("g1", "g2", "g3")
res <- computeVSIF(freq = freq.vec, n, sigma.sq)

freq.mat <- matrix(c(0.1, 0.2, 0.5, 0.1, 0.01, 0.5), nrow = 2, byrow = TRUE)
colnames(freq.mat) <- names(sigma.sq)
res <- computeVSIF(freq = freq.mat, n, sigma.sq)


library(GWASTools)
n <- 1000
set.seed(22)
outcome <- c(rnorm(n*0.28, sd =1), rnorm(n*0.7, sd = 1), rnorm(n*0.02, sd = sqrt(2)) )
dat <- data.frame(sample.id=paste0("ID_", 1:n),
                  outcome = outcome,
                  b=c(rep("g1", n*0.28), rep("g2", n*0.7), rep("g3", n*0.02)),
                  stringsAsFactors=FALSE)
dat <- AnnotatedDataFrame(dat)
nm <- fitNullModel(dat, outcome="outcome", covars="b", verbose=FALSE)
freq.vec <- c(0.1, 0.2, 0.5)
names(freq.vec) <- c("g1", "g2", "g3")
group.var.vec <- dat$b
names(group.var.vec) <- dat$sample.id

res <- computeVSIFNullModel(nm, freq.vec, group.var.vec)

freq.mat <- matrix(c(0.1, 0.2, 0.5, 0.1, 0.01, 0.5), nrow = 2, byrow = TRUE)
colnames(freq.mat) <- c("g1", "g2", "g3")
```

```
res <- computeVSIFNullModel(nm, freq.mat, group.var.vec)
```

`effectAllele` *Return the effect allele for association testing*

Description

`effectAllele` returns the effect allele for association testing.

Usage

```
## S4 method for signature 'SeqVarGDSClass'
effectAllele(gdsobj, variant.id=NULL)
## S4 method for signature 'GenotypeData'
effectAllele(gdsobj, variant.id=NULL)
```

Arguments

`gdsobj`	An object of class `SeqVarIterator` from the package **SeqVarTools**, or an object of class `GenotypeIterator` from the package **GWASTools**, containing the genotype data for the variants and samples to be used for the analysis.
`variant.id`	A vector of identifiers for variants to return.

Details

`effectAllele` returns the effect allele corresponding to association test results from `assocTestSingle` or `assocTestAggregate`. `variant.id` allows the user to specify for which variants effect alleles should be returned.

Value

A data.frame with the following columns:

`variant.id`	The variant ID
`effect.allele`	The character value for the effect allele
`other.allele`	The character value for the other (non-effect) allele

Author(s)

Stephanie M. Gogarten

See Also

`assocTestSingle`, `assocTestAggregate`

Examples

```
library(SeqVarTools)
library(Biobase)

# open a sequencing GDS file
gdsfile <- seqExampleFileName("gds")
gds <- seqOpen(gdsfile)

# simulate some phenotype data
set.seed(4)
data(pedigree)
pedigree <- pedigree[match(seqGetData(gds, "sample.id"), pedigree$sample.id),]
pedigree$outcome <- rnorm(nrow(pedigree))

# construct a SeqVarIterator object
seqData <- SeqVarData(gds, sampleData=AnnotatedDataFrame(pedigree))
iterator <- SeqVarBlockIterator(seqData)

# fit the null model
nullmod <- fitNullModel(iterator, outcome="outcome", covars="sex")

# run the association test
assoc <- assocTestSingle(iterator, nullmod)

# add effect allele to the results
eff <- effectAllele(seqData, variant.id=assoc$variant.id)
assoc <- dplyr::left_join(assoc,eff)
head(assoc)

seqClose(iterator)
```

`fitNullModel` *Fit a Model Under the Null Hypothesis*

Description

`fitNullModel` fits a regression model or a mixed model with random effects specified by their covariance structures; this allows for the inclusion of a polygenic random effect using a kinship matrix or genetic relationship matrix (GRM). The output of `fitNullModel` can be used to estimate genetic heritability and can be passed to `assocTestSingle` or `assocTestAggregate` for the purpose of genetic association testing.

`nullModelInvNorm` does an inverse normal transform of a previously fit null model.

`nullModelSmall` returns a small version of the null model with no NxN matrices.

`isNullModelSmall` returns TRUE if a null model is small; FALSE otherwise.

`fitNullModelFastScore` fits a null model that can be used for association testing with the fast approximation to the score standard error (SE).

`calcScore` calculates the score, its true SE, and the fast SE for a set of variants; used to compute the SE correction factor used for the fast approximation.

`nullModelFastScore` updates a previously fit null model so that it can be used for association testing with the fast approximation to the score SE.

`isNullModelFastScore` returns TRUE if a null model can be used for association testing with the fast approximation to the score SE; FALSE otherwise.

Usage

```
## S4 method for signature 'data.frame'
fitNullModel(x, outcome, covars = NULL, cov.mat = NULL,
            group.var = NULL, family = "gaussian", two.stage = FALSE,
        norm.option = c("all", "by.group"), rescale = c("residSD", "none", "model"),
            start = NULL, AIREML.tol = 1e-4, max.iter = 100, EM.iter = 0,
            drop.zeros = TRUE, return.small = FALSE, verbose = TRUE)
## S4 method for signature 'AnnotatedDataFrame'
fitNullModel(x, outcome, covars = NULL, cov.mat = NULL,
            group.var = NULL, sample.id = NULL, ...)
## S4 method for signature 'SeqVarData'
fitNullModel(x, ...)
## S4 method for signature 'ScanAnnotationDataFrame'
fitNullModel(x, ...)
## S4 method for signature 'GenotypeData'
fitNullModel(x, ...)

nullModelInvNorm(null.model, cov.mat = NULL,
                 norm.option = c("all", "by.group"),
                 rescale = c("residSD", "none", "model"),
                 AIREML.tol = 1e-4, max.iter = 100, EM.iter = 0,
                 drop.zeros = TRUE, return.small = FALSE, verbose = TRUE)

nullModelSmall(null.model)

isNullModelSmall(null.model)

## S4 method for signature 'SeqVarData'
fitNullModelFastScore(x, outcome, covars = NULL, cov.mat = NULL,
            group.var = NULL, family = "gaussian", two.stage = FALSE,
        norm.option = c("all", "by.group"), rescale = c("residSD", "none", "model"),
            start = NULL, AIREML.tol = 1e-4, max.iter = 100, EM.iter = 0,
            drop.zeros = TRUE, return.small = TRUE,
            variant.id = NULL, nvar = 100, min.mac = 20, sparse = TRUE,
          imputed = FALSE, male.diploid = TRUE, genome.build = c("hg19", "hg38"),
            verbose = TRUE)

calcScore(x, null.model, variant.id = NULL, nvar = 100, min.mac = 20, sparse = TRUE,
         imputed = FALSE, male.diploid = TRUE, genome.build = c("hg19", "hg38"),
          verbose = TRUE)
```

```
nullModelFastScore(null.model, score.table, return.small = TRUE, verbose = TRUE)

isNullModelFastScore(null.model)
```

Arguments

x	An object of class data.frame, AnnotatedDataFrame, or SeqVarData containing the outcome and covariate data for the samples to be used for the analysis. Must be class SeqVarData when using fitNullModelFastScore or calcScore. See 'Details' for more information.
outcome	A character string specifying the name of the outcome variable in x.
covars	A vector of character strings specifying the names of the fixed effect covariates in x; an intercept term is automatically included. If NULL (default), the only fixed effect covariate is the intercept term.
cov.mat	A matrix or list of matrices specifying the covariance structures of the random effects terms. Objects from the **Matrix** package are supported. If NULL (default), no random effects terms are included. See 'Details' for more information.
group.var	This variable can only be used when family = "gaussian". A character string specifying the name of a categorical variable in x that is used to fit heterogeneous residual error variances. If NULL (default), then a standard LMM with constant residual variance for all samples is fit. See 'Details' for more information.
sample.id	A vector of IDs for samples to include in the analysis. If NULL, all samples in x are included. This argument is ignored if x is a data.frame; see 'Details'.
family	A description of the error distribution to be used in the model. The default "gaussian" fits a linear model; "binomial" and "poisson" are also supported. See 'Details' for more information.
two.stage	Logical indicator of whether to use a fully-adjusted two-stage rank normalization procedure for fitting the model. Can only be used when family = "gaussian". See 'Details' for more information.
norm.option	Specifies whether the rank normalization should be done separately within each value of group.var ("by.group") or with all samples together ("all") when using two.stage or nullModelInvNorm.
rescale	Specifies how to rescale the residuals after rank normalization when using two.stage or nullModelInvNorm.: "none" for no rescaling of the residuals; "model" to rescale by the model-based variance components, and "residSD" (default) to rescale by the standard deviation of the original marginal residuals. Rescaling is done with the same grouping as the rank normalization, as specified by norm.option.
start	A vector of starting values for the variance component estimation procedure. The function will pick reasonable starting values when left NULL (default). See 'Details' for more information.
AIREML.tol	The convergence threshold for the Average Information REML (AIREML) procedure used to estimate the variance components of the random effects. See 'Details' for more information.

max.iter	The maximum number of iterations allowed to reach convergence.
EM.iter	The number of EM iterations to run prior to AIREML; default is 0.
drop.zeros	Logical indicator of whether variance component terms that converge to 0 should be removed from the model; the default is TRUE. See 'Details' for more information.
return.small	Logical for whether to return a small version of the null model without NxN matrices. Default for fitNullModel is FALSE; only set to TRUE for use in association tests with test = "BinomiRare" or test = "CMP" and recalc.pval.thresh = 1. Default for fitNullModelFastScore is TRUE, as NxN matrices are not needed for the fast score SE approximation.
null.model	The output of fitNullModel.
variant.id	Optional list of variant.ids in x specifying which variants to use for computing the SE correction factor; if NULL, a random selection of nvar variants with minor allele count at least min.mac is used.
nvar	The number of random variants to select from x for computing the SE correction factor; ignored if variant.id is specified.
min.mac	The minimum minor allele count allowed for the random variants selected from x for computing the SE correction factor; ignored if variant.id is specified.
sparse	Logical indicator of whether to read genotypes as sparse Matrix objects; the default is TRUE. Set this to FALSE if the alternate allele dosage of the genotypes in the test are not expected to be mostly 0.
imputed	Logical indicator of whether to read dosages from the "DS" field containing imputed dosages instead of counting the number of alternate alleles.
male.diploid	Logical for whether males on sex chromosomes are coded as diploid.
genome.build	A character sting indicating genome build; used to identify pseudoautosomal regions on the X and Y chromosomes.
verbose	Logical indicator of whether updates from the function should be printed to the console; the default is TRUE.
...	Arguments to pass to other methods.
score.table	The output of calcScore.

Details

If x is a data.frame, the rownames of x must match the row and column names of cov.mat (if cov.mat is specified). If x is an AnnotatedDataFrame or other object containing an AnnotatedDataFrame, x will be re-ordered (if necessary) so that sample.id or scanID is in the same order as the row and column names of cov.mat.

If any covariates have the same value for all samples, they will be dropped from the model with a message. Note that the 'model' and 'covars' element in the output object will still include that covariate.

The code checks for multicollinearity of covariates by checking that the rank of the design matrix is equal to the number of columns; if the rank is smaller, it fails with an error.

cov.mat is used to specify the covariance structures of the random effects terms in the model. For example, to include a polygenic random effect, one matrix in cov.mat could be a kinship matrix

or a genetic relationship matrix (GRM). As another example, to include household membership as a random effect, one matrix in cov.mat should be a 0/1 matrix with a 1 in the [i,j] and [j,i] entries if individuals i and j are in the same household and 0 otherwise; the diagonals of such a matrix should all be 1. If cov.mat is a list without names, the code will assign sequential letters as names. If some elements are named but not others, it will produce an error.

For some outcomes, there may be evidence that different groups of observations have different residual variances, and the standard LMM assumption of homoscedasticity is violated. When group.var is specified, separate (heterogeneous) residual variance components are fit for each unique value of group.var. This parameter is only applicable when family = "gaussian".

When family is not gaussian, the penalized quasi-likelihood (PQL) approximation to the generalized linear mixed model (GLMM) is fit following the procedure of GMMAT (Chen et al., 2016).

Let m be the number of matrices in cov.mat and let g be the number of categories in the variable specified by group.var. The length of the start vector must be (m + 1) when family is gaussian and group.var is NULL; (m + g) when family is gaussian and group.var is specified; or m when family is not gaussian.

A Newton-Raphson iterative procedure with Average Information REML (AIREML) is used to estimate the variance components of the random effects. When the absolute change between all of the new and previous variance component estimates is less than var(outcome)*AIREML.tol, the algorithm declares convergence of the estimates. Sometimes a variance component may approach the boundary of the parameter space at 0; step-halving is used to prevent any component from becomming negative. However, when a variance component gets near the 0 boundary, the algorithm can sometimes get "stuck", preventing the other variance components from converging; if drop.zeros is TRUE, then variance components that converge to a value less than AIREML.tol will be dropped from the model and the estimation procedure will continue with the remaining variance components.

When two.stage = TRUE, the fully-adjusted two-stage rank normalization procedure from Sofer et. al. (2019) is used. With this procedure: the stage 1 model is fit as usual, with the specified outcome, covars, cov.mat, and group.var; the marginal residuals from the stage 1 model are rank normalized as specified by norm.option and then rescaled as specified by rescale; the stage 2 model is then fit using the rank normalized and rescaled residuals as the new outcome, with the same covars, cov.mat, and group.var as the stage 1 model. The output of this stage 2 model is saved and should be used for association testing. This procedure is only applicable when family = "gaussian". The nullModelInvNorm function takes a previously fit null model as input and does the rank normalization, rescaling, and stage 2 model fitting.

The function fitNullModelFastScore fits a null model that can be used for testing variant association with the fast approximation to the score standard error (SE) implemented by Zhou et al. (2018) in their SAIGE software. This approximation may be much faster than computing the true score SE in large samples, as it replaces the full covariance matrix in the calculation with the product of a diagonal matrix and a scalar correction factor (se.correction in the null model output); see the option fast.Score.SE in assocTestSingle for further details. A null model previously fit with fitNullModel can be updated to this format by using calcScore to compute the necessary statistics on a set of variants, followed by nullModelFastScore to calculate the se.correction factor and update the null model format.

Value

An object of class 'GENESIS.nullModel' or 'GENESIS.nullMixedModel'. A list including:

model A list with elements describing the model that was fit. See below for explanations of the elements in this list.

varComp The variance component estimates. There is one variance component for each random effect specified in `cov.mat`. When `family` is gaussian, there are additional residual variance components; one residual variance component when `group.var` is `NULL`, and as many residual variance components as there are unique values of `group.var` when it is specified.

varCompCov The estimated covariance matrix of the variance component estimates given by `varComp`. This can be used for hypothesis tests regarding the variance components.

fixef A data.frame with effect size estimates (betas), standard errors, chi-squared test statistics, and p-values for each of the fixed effect covariates specified in `covars`.

betaCov The estimated covariance matrix of the effect size estimates (betas) of the fixed effect covariates. This can be used for hypothesis tests regarding the fixed effects.

fit A data frame with the outcome, fitted values, and residuals. See below for explanations of the columns of this data frame.

logLik The log-likelihood value.

logLikR The restricted log-likelihood value.

AIC The Akaike Information Criterion value.

model.matrix The design matrix for the fixed effect covariates used in the model.

group.idx If `group.var` is not `NULL`, a list of indices for samples in each group.

cholSigmaInv The Cholesky decomposition of the inverse of the estimated outcome covariance structure. This is used by `assocTestSingle` or `assocTestAggregate` for genetic association testing.

W The diagonal weight matrix with terms given by the variance function for the specified `family`. This is the inverse of portion of the estimated outcome covariance structure not attributed to random effects specified with `cov.mat`. This matrix is used in place of the inverse of the estimated outcome covariance structure when using `fast.score.SE` for association testing with `assocTestSingle`.

converged A logical indicator of whether the AIREML procedure for estimating the random effects variance components converged.

zeroFLAG A vector of logicals the same length as `varComp` specifying whether the corresponding variance component estimate was set to 0 by the function due to convergence to the boundary in the AIREML procedure.

RSS The residual sum of squares from the model fit. When `family` is gaussian, this will typically be 1 since the residual variance component is estimated separately.

RSS0 the sum of resid.cholesky squared. This is the sum of squared residuals under the null hypothesis of no genetic effect for the covariate and random effect adjusted model using the Frisch-Waugh-Lovell theorem.

CX a matrix needed for calculating association test statistics

CXCXI a matrix needed for calculating association test statistics

se.correction The scalar correction factor for the fast approximation to the score SE; the average of the `se.ratio` values in `score.table`.

score.table A data frame with information about the variants used to compute the `se.correction` factor.

The `fit` data frame contains the following columns, depending on which type of model was fit:

outcome The original outcome vector.

workingY The "working" outcome vector. When `family` is gaussian, this is just the original outcome vector. When `family` is not gaussian, this is the PQL linearization of the outcome vector. This is used by `assocTestSingle` or `assocTestAggregate` for genetic association testing. See 'Details' for more information.

fitted.values The fitted values from the model. For mixed models, this is `X*beta` where `X` is the design matrix and beta is the vector of effect size estimates for the fixed effects. For non-mixed models, this is the inverse link function applied to `X*beta` (e.g., `expit(X*beta)` for logistic regression when `family = "binomial"`).

resid.marginal The marginal residuals from the model; i.e. Y - X*beta where Y is the vector of outcome values.

linear.predictor The linear predictor from the model; i.e. X*beta + Z*u, where Z*u represents the effects captured by the random effects specified with the cov.mat input.

resid.conditional The conditional residuals from the model; i.e. Y - X*beta - Z*u, where Z*u represents the effects captured by the random effects specified with the cov.mat input.

resid.cholesky The Cholesky residuals from the model; a transformation of the marginal residuals using the estimated model covariance structure such that they are uncorrelated and follow a standard multivariate Normal distribution with mean 0 and identity covariance matrix asymptotically. Linear regression of the Cholesky residual vector on an equivalently transformed genotype vector provides the same estimates as fitting the full GLS model (by the Frisch-Waugh-Lovell theorem).

resid.PY The outcome vector (Y) pre-multiplied by a projection matrix (P) that adjusts for covariates, random effects, and model covariance structure. These projected outcome values are essentially what are correlated with genotype values for association testing.

sample.id A vector of IDs for the samples used in the analysis, if called with an `AnnotatedDataFrame`.

The `model` list element contains the following elements:

outcome The outcome variable name.

covars A vector of covariate names

formula The model string.

family A character string specifying the family used in the analysis.

hetResid A logical indicator of whether heterogeneous residual variance components were used in the model (specified by `group.var`).

The `score.table` data frame contains the following columns:

variant.id The variant ID

chr The chromosome value

pos The base pair position

allele.index The index of the alternate allele. For biallelic variants, this will always be 1.

n.obs The number of samples with non-missing genotypes

freq The estimated alternate allele frequency

MAC The minor allele count. For multiallelic variants, "minor" is determined by comparing the count of the alternate allele specified by `allele.index` with the sum of all other alleles.

Score The value of the score function

Score.SE The estimated true standard error of the Score

Score.SE.fast The estimated fast standard error of the Score (before scalar correction)

se.ratio The ratio of Score.SE to Score.SE.fast; these values are averaged across varaints to estimate `se.correction` in `nullModelFastScore`.

Author(s)

Matthew P. Conomos, Stephanie M. Gogarten, Tamar Sofer, Ken Rice, Chaoyu Yu

References

Chen H, Wang C, Conomos MP, Stilp AM, Li Z, Sofer T, Szpiro AA, Chen W, Brehm JM, Celedon JC, Redline S, Papanicolaou GJ, Thornton TA, Laurie CC, Rice K and Lin X. (2016) Control for Population Structure and Relatedness for Binary Traits in Genetic Association Studies Using Logistic Mixed Models. American Journal of Human Genetics, 98(4):653-66.

Sofer, T., Zheng, X., Gogarten, S. M., Laurie, C. A., Grinde, K., Shaffer, J. R., ... & Rice, K. M. (2019). A fully adjusted two-stage procedure for rank-normalization in genetic association studies. Genetic epidemiology, 43(3), 263-275.

Zhou, W., Nielsen, J. B., Fritsche, L. G., Dey, R., Gabrielsen, M. E., Wolford, B. N., ... & Bastarache, L. A. (2018). Efficiently controlling for case-control imbalance and sample relatedness in large-scale genetic association studies. Nature genetics, 50(9), 1335.

Breslow NE and Clayton DG. (1993). Approximate Inference in Generalized Linear Mixed Models. Journal of the American Statistical Association 88: 9-25.

Gilmour, A.R., Thompson, R., & Cullis, B.R. (1995). Average information REML: an efficient algorithm for variance parameter estimation in linear mixed models. Biometrics, 1440-1450.

See Also

`varCompCI` for estimating confidence intervals for the variance components and the proportion of variability (heritability) they explain, `assocTestSingle` or `assocTestAggregate` for running genetic association tests using the output from `fitNullModel`.

Examples

```
library(GWASTools)

# file path to GDS file
gdsfile <- system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS")
# read in GDS data
HapMap_geno <- GdsGenotypeReader(filename = gdsfile)
# create a GenotypeData class object
HapMap_genoData <- GenotypeData(HapMap_geno)
# load saved matrix of KING-robust estimates
data("HapMap_ASW_MXL_KINGmat")
```

```
# run PC-AiR
mypcair <- pcair(HapMap_genoData, kinobj = HapMap_ASW_MXL_KINGmat,
                 divobj = HapMap_ASW_MXL_KINGmat)

# run PC-Relate
HapMap_genoData <- GenotypeBlockIterator(HapMap_genoData, snpBlock=20000)
mypcrel <- pcrelate(HapMap_genoData, pcs = mypcair$vectors[,1,drop=FALSE],
     training.set = mypcair$unrels,
                        BPPARAM = BiocParallel::SerialParam())
close(HapMap_genoData)

# generate a phenotype
set.seed(4)
pheno <- 0.2*mypcair$vectors[,1] + rnorm(mypcair$nsamp, mean = 0, sd = 1)

annot <- data.frame(sample.id = mypcair$sample.id,
                    pc1 = mypcair$vectors[,1], pheno = pheno)

# make covariance matrix
cov.mat <- pcrelateToMatrix(mypcrel, verbose=FALSE)[annot$sample.id, annot$sample.id]

# fit the null mixed model
nullmod <- fitNullModel(annot, outcome = "pheno", covars = "pc1", cov.mat = cov.mat)
```

GENESIS-defunct *Defunct functions in package* **GENESIS**

Description

These functions are defunct and no longer available.

Details

The following functions are defunct; use the replacement indicated below:

- admixMapMM: admixMap
- assocTestMM: assocTestSingle
- assocTestSeq: assocTestAggregate
- assocTestSeqWindow: assocTestAggregate
- fitNullMM: fitNullModel
- fitNullReg: fitNullModel
- king2mat: kingToMatrix
- pcrelate,GenotypeData-method: pcrelate,GenotypeIterator-method
- pcrelate,SeqVarData-method: pcrelate,SeqVarIterator-method
- pcrelateMakeGRM: pcrelateToMatrix
- pcrelateReadInbreed: pcrelate
- pcrelateReadKinship: pcrelate

HapMap_ASW_MXL_KINGmat

Matrix of Pairwise Kinship Coefficient Estimates for the combined HapMap ASW and MXL Sample found with the KING-robust estimator from the KING software.

Description

KING-robust kinship coefficient estimates for the combined HapMap African Americans in the Southwest U.S. (ASW) and Mexican Americans in Los Angeles (MXL) samples.

Usage

```
data(HapMap_ASW_MXL_KINGmat)
```

Format

The format is: num [1:173, 1:173] 0 0.00157 -0.00417 0.00209 0.00172 ...

Value

A matrix of pairwise kinship coefficient estimates as calculated with KING-robust for the combined HapMap African Americans in the Southwest U.S. (ASW) and Mexican Americans in Los Angeles (MXL) samples.

Source

http://hapmap.ncbi.nlm.nih.gov/

References

International HapMap 3 Consortium. (2010). Integrating common and rare genetic variation in diverse human populations. Nature, 467(7311), 52-58.

jointScoreTest *Perform a joint score test*

Description

`jointScoreTest` is used to perform a joint score test of a set of variants using a null model and a matrix of genotype dosages.

Usage

```
jointScoreTest(null.model, G)
```

Arguments

null.model	A null model object returned by fitNullModel.
G	A matrix of genotype dosages, where samples are the rows and variants are the columns.

Details

jointScoreTest takes the object returned by fitNullModel and performs a joint score test for all variants in the G matrix using this null model. All effect size and PVE estimates in fixef are adjusted for (i.e. conditional on) all other variants included in G.

The G matrix must be formatted such that the rows are samples and the columns are variants, and the entries are the dosage for that sample and variant. The matrix must have sample.id as rownames; this is used to match the genotypes to the null model. Therefore, the same sample identifiers must be used in both null.model and G. G can contain additional samples and ordering is unimportant as long as all samples from the null model are present; it will be subset and reordered to match the set of samples in the null model.

Colnames for G are not required. If present, the column names of G will be used as the rownames of the fixef element and the column and rownames of the betaCov element of the output. fixef and betaCov will be in the same order as the columns in G.

Missing data in G are not allowed.

Value

jointScoreTest returns a list with the following elements:

Joint.Stat	Squared joint score test statistic for all variants in G.
Joint.pval	p-value associated with the joint score test statistic drawn from a χ^2 distribution with n_{variants} degrees of freedom.
Joint.PVE	Estimate of the proportion of variance explained jointly by the variants in G.
fixef	A data.frame with joint effect size estimates (Est), standard errors (SE), chi-squared test statistics (Stat), p-values (pval), and estimated proportion of variance explained (PVE) for each of the variants specified in G.
betaCov	Estimated covariance matrix for the variants in G.

Author(s)

Adrienne M. Stilp, Matthew P. Conomos

See Also

fitNullModel for fitting the mixed model and performing the variance component estimation. GenotypeData and SeqVarData for obtaining genotype matrices.

Examples

```
library(GWASTools)

# File path to GDS file
gdsfile <- system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS")
# Read in GDS data
HapMap_geno <- GdsGenotypeReader(filename = gdsfile)
# Create a GenotypeData class object
HapMap_genoData <- GenotypeData(HapMap_geno)

# Prepare genotypes for the first five SNPs and the first 20 samples.
# Transpose it so that samples are rows and SNPs are columns.
geno <- t(getGenotype(HapMap_genoData, snp = c(1, 5), scan = c(1, 20)))
# Set row and column names.
rownames(geno) <- as.character(GWASTools::getScanID(HapMap_genoData))[1:20]
colnames(geno) <- sprintf("snp%s", 1:5)

# Create a phenotype
set.seed(5)
phen <- data.frame(
  outcome = rnorm(1:20),
  covar = rnorm(1:20),
  stringsAsFactors = FALSE
)
rownames(phen) <- rownames(geno)

# load saved matrix of KING-robust estimates
data("HapMap_ASW_MXL_KINGmat")
cov.mat = HapMap_ASW_MXL_KINGmat[rownames(phen), rownames(phen)]

# Fit a null model.
nullmod <- fitNullModel(phen,
                        outcome = "outcome",
                        covars = "covar",
                        cov.mat = cov.mat)


# Run the joint score test.
jointScoreTest(nullmod, geno)

close(HapMap_genoData)
```

---

kin2gds *Store kinship matrix in GDS*

Description

kin2gds and mat2gds write kinship matrices to GDS files.

Usage

```
kin2gds(ibdobj, gdsfile)
mat2gds(mat, gdsfile)
```

Arguments

ibdobj	A list with elements sample.id and kinship, such as the output of snpgdsIBDKING.
mat	A matrix with sample identifiers in colnames.
gdsfile	A character string with the name of the GDS file to create.

Details

When using pcair or pcairPartition with large sample sizes, it can be more memory efficient to read data from GDS files. kin2gds and mat2gds store kinship matrices in GDS files for use with these functions.

Author(s)

Stephanie M. Gogarten

See Also

kingToMatrix, snpgdsIBDKING

Examples

```
library(gdsfmt)

# KING results from the command-line program
file.kin0 <- system.file("extdata", "MXL_ASW.kin0", package="GENESIS")
file.kin <- system.file("extdata", "MXL_ASW.kin", package="GENESIS")
KINGmat <- kingToMatrix(c(file.kin0, file.kin), estimator="Kinship")
gdsfile <- tempfile()
mat2gds(KINGmat, gdsfile)
gds <- openfn.gds(gdsfile)
gds
closefn.gds(gds)

# KING results from SNPRelate
library(SNPRelate)
geno <- snpgdsOpen(snpgdsExampleFileName())
king <- snpgdsIBDKING(geno)
closefn.gds(geno)
kin2gds(king, gdsfile)
gds <- openfn.gds(gdsfile)
gds
closefn.gds(gds)
```

kingToMatrix *Convert KING text output to an R Matrix*

Description

kingToMatrix is used to extract the pairwise kinship coefficient estimates from the output text files of KING –ibdseg, KING –kinship, or KING –related and put them into an R object of class Matrix. One use of this matrix is that it can be read by the functions pcair and pcairPartition.

Usage

```
## S4 method for signature 'character'
kingToMatrix(king, estimator = c("PropIBD", "Kinship"), sample.include = NULL,
    thresh = NULL, verbose = TRUE)
## S4 method for signature 'snpgdsIBDClass'
kingToMatrix(king, sample.include = NULL, thresh = 2^(-11/2), verbose = TRUE)
```

Arguments

king	Output from KING, either a snpgdsIBDClass object from snpgdsIBDKING or a character vector of one or more file names output from the command-line version of KING; see 'Details'.
estimator	Which estimates to read in when using command-line KING output; must be either "PropIBD" or "Kinship"; see 'Details'.
sample.include	An optional vector of sample.id indicating all samples that should be included in the output matrix; see 'Details' for usage.
thresh	Kinship threshold for clustering samples to make the output matrix sparse block-diagonal. When NULL, no clustering is done. See 'Details'.
verbose	A logical indicating whether or not to print status updates to the console; the default is TRUE.

Details

king can be a vector of multiple file names if your KING output is stored in multiple files; e.g. KING –kinship run with family IDs returns a .kin and a .kin0 file for pairs within and not within the same family, respectively.

When reading command-line KING output, the estimator argument is required to specify which estimates to read in. When reading KING –ibdseg output, only "PropIBD" will be available; when reading KING –kinship output, only "Kinship" will be available; when reading KING –related output, both "PropIBD" and "Kinship" will be available - use this argument to select which to read. See the KING documentation for details on each estimator.

sample.include has two primary functions: 1) It can be used to subset the KING output. 2) sample.include can include sample.id not in king; this ensures that all samples will be in the output matrix when reading KING –ibdseg output, which likely does not contain all pairs. When left NULL, the function will determine the list of samples from what is observed in king. It is

recommended to use sample.include to ensure all of your samples are included in the output matrix.

thresh sets a threhsold for clustering samples such that any pair with an estimated kinship value greater than thresh is in the same cluster. All pairwise estimates within a cluster are kept, even if they are below thresh. All pairwise estimates between clusters are set to 0, creating a sparse, block-diagonal matrix. When thresh is NULL, no clustering is done and all samples are returned in one block. This feature is useful when converting KING –ibdseg or KING –robust estimates to be used as a kinship matrix, if you have a lower threshold that you consider 'related'. This feature should not be used when converting KING –robust estimates to be used as divobj in pcair or pcairPartition, as PC-AiR requires the negative estimates to identify ancestrally divergent pairs.

Value

An object of class 'Matrix' with pairwise kinship coefficients by KING –ibdseg or KING –robust for each pair of individuals in the sample. The estimates are on both the upper and lower triangle of the matrix, and the diagonal is arbitrarily set to 0.5. sample.id are set as the column and row names of the matrix.

Author(s)

Matthew P. Conomos

References

Conomos M.P., Miller M., & Thornton T. (2015). Robust Inference of Population Structure for Ancestry Prediction and Correction of Stratification in the Presence of Relatedness. Genetic Epidemiology, 39(4), 276-293.

Manichaikul, A., Mychaleckyj, J.C., Rich, S.S., Daly, K., Sale, M., & Chen, W.M. (2010). Robust relationship inference in genome-wide association studies. Bioinformatics, 26(22), 2867-2873.

See Also

pcair and pcairPartition for functions that use the output matrix.

Examples

```
# KING --kinship
file.king <- c(system.file("extdata", "MXL_ASW.kin0", package="GENESIS"),
               system.file("extdata", "MXL_ASW.kin", package="GENESIS"))
KINGmat <- kingToMatrix(file.king, estimator="Kinship")

# KING --ibdseg
file.king <- system.file("extdata", "HapMap.seg", package="GENESIS")
KINGmat <- kingToMatrix(file.king, estimator="PropIBD")

# SNPRelate
library(SNPRelate)
gds <- snpgdsOpen(system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS"))
king <- snpgdsIBDKING(gds)
KINGmat <- kingToMatrix(king)
```

```
snpgdsClose(gds)
```

`makeSparseMatrix` *Make a sparse matrix from a dense matrix or a table of pairwise values*

Description

`makeSparseMatrix` is used to create a sparse block-diagonal matrix from a dense matrix or a table of pairwise values, using a user-specified threshold for sparsity. An object of class `Matrix` will be returned. A sparse matrix may be useful for fitting the association test null model with `fitNullModel` when working with very large sample sizes.

Usage

```
## S4 method for signature 'data.table'
makeSparseMatrix(x, thresh = NULL, sample.include = NULL, diag.value = NULL,
                 verbose = TRUE)
## S4 method for signature 'data.frame'
makeSparseMatrix(x, thresh = NULL, sample.include = NULL, diag.value = NULL,
                 verbose = TRUE)
## S4 method for signature 'matrix'
makeSparseMatrix(x, thresh = 2^(-11/2), sample.include = NULL, diag.value = NULL,
                 verbose = TRUE)
## S4 method for signature 'Matrix'
makeSparseMatrix(x, thresh = 2^(-11/2), sample.include = NULL, diag.value = NULL,
                 verbose = TRUE)
```

Arguments

`x`	An object to coerce to a sparse matrix. May be of class `matrix`, `Matrix`, `data.frame`, or `data.table`. When a `matrix` or `Matrix`, row and column names should be set to sample.ids; when a data.frame or data.table, should have 3 columns: ID1, ID2, and value.
`thresh`	Value threshold for clustering samples to make the output matrix sparse block-diagonal. When `NULL`, no clustering is done. See 'Details'.
`sample.include`	An optional vector of sample.id indicating all samples that should be included in the output matrix; see 'Details' for usage.
`diag.value`	When `NULL` (by Default), values for the diagonal of the output matrix should be provided in `x`. Setting `diag.value` to a numeric value will make all values on the diagonal of the output matrix that value.
`verbose`	A logical indicating whether or not to print status updates to the console; the default is TRUE.

Details

`sample.include` has two primary functions: 1) It can be used to subset samples provided in `x`. 2) `sample.include` can include sample.id not in `x`; these additional samples will be included in the output matrix, with a value of 0 for all off-diagonal elements, and the value provided by `diag.value` for the diagonal elements. When left `NULL`, the function will determine the list of samples from what is observed in `x`.

`thresh` sets a threhsold for clustering samples such that any pair with a value greater than `thresh` is in the same cluster. All values within a cluster are kept, even if they are below `thresh`. All values between clusters are set to 0, creating a sparse, block-diagonal matrix. When `thresh` is `NULL`, no clustering is done and all samples are returned in one block. This feature is useful when converting a data.frame of kinship estimates to a matrix.

Value

An object of class '`Matrix`'. Samples may be in a different order than in the input `x`, as samples are sorted by ID or rowname within each block (including within the block of unrelateds).

Author(s)

Matthew P. Conomos

See Also

`kingToMatrix` and `pcrelateToMatrix` for functions that use this function.

`pcair` *PC-AiR: Principal Components Analysis in Related Samples*

Description

`pcair` is used to perform a Principal Components Analysis using genome-wide SNP data for the detection of population structure in a sample. Unlike a standard PCA, PC-AiR accounts for sample relatedness (known or cryptic) to provide accurate ancestry inference that is not confounded by family structure.

Usage

```
## S4 method for signature 'gds.class'
pcair(gdsobj, kinobj = NULL, divobj = NULL,
                            kin.thresh = 2^(-11/2), div.thresh = -2^(-11/2),
                            unrel.set = NULL,
                            sample.include = NULL, snp.include = NULL,
                            num.cores = 1L, verbose = TRUE, ...)
## S4 method for signature 'SNPGDSFileClass'
pcair(gdsobj, ...)
## S4 method for signature 'GdsGenotypeReader'
pcair(gdsobj, ...)
```

```
## S4 method for signature 'MatrixGenotypeReader'
pcair(gdsobj, ...)
## S4 method for signature 'GenotypeData'
pcair(gdsobj, ...)
## S4 method for signature 'SeqVarGDSClass'
pcair(gdsobj, ...)
```

Arguments

gdsobj	An object providing a connection to a GDS file.
kinobj	A symmetric matrix of pairwise kinship coefficients for every pair of individuals in the sample: upper and lower triangles must both be filled; diagonals should be self-kinship or set to a non-missing constant value. This matrix is used for partitioning the sample into the 'unrelated' and 'related' subsets. See 'Details' for how this interacts with `kin.thresh` and `unrel.set`. IDs for each individual must be set as the column names of the matrix. This matrix may also be provided as a GDS object; see 'Details'.
divobj	A symmetric matrix of pairwise ancestry divergence measures for every pair of individuals in the sample: upper and lower triangles must both be filled; diagonals should be set to a non-missing constant value. This matrix is used for partitioning the sample into the 'unrelated' and 'related' subsets. See 'Details' for how this interacts with `div.thresh`. IDs for each individual must be set as the column names of the matrix. This matrix may be identical to `kinobj`. This matrix may be `NULL` to ignore ancestry divergence. This matrix may also be provided as a GDS object; see 'Details'.
kin.thresh	Threshold value on `kinobj` used for declaring each pair of individuals as related or unrelated. The default value is 2^(-11/2) ~ 0.022, corresponding to 4th degree relatives. See 'Details' for how this interacts with `kinobj`.
div.thresh	Threshold value on `divobj` used for deciding if each pair of individuals is ancestrally divergent. The default value is -2^(-11/2) ~ -0.022. See 'Details' for how this interacts with `divobj`.
unrel.set	An optional vector of IDs for identifying individuals that are forced into the unrelated subset. See 'Details' for how this interacts with `kinobj`.
sample.include	An optional vector of IDs for selecting samples to consider for either set.
snp.include	An optional vector of snp or variant IDs to use in the analysis.
num.cores	The number of cores to use.
verbose	Logical indicator of whether updates from the function should be printed to the console; the default is TRUE.
...	Additional arguments to pass to `snpgdsPCA`, such as `eigen.cnt` to control the number of PCs returned, `num.thread` for parallel execution, or `algorithm='randomized'` for `fastPCA` (recommended for large sample sets).

Details

The basic premise of PC-AiR is to partition the entire sample of individuals into an ancestry representative 'unrelated subset' and a 'related set', perform standard PCA on the 'unrelated subset', and predict PC values for the 'related subset'.

We recommend using software that accounts for population structure to estimate pairwise kinship coefficients to be used in kinobj. Any pair of individuals with a pairwise kinship greater than kin.thresh will be declared 'related.' Kinship coefficient estimates from the KING-robust software are typically used as measures of ancestry divergence in divobj. Any pair of individuals with a pairwise divergence measure less than div.thresh will be declared ancestrally 'divergent'. Typically, kin.thresh and div.thresh are set to be the amount of error around 0 expected in the estimate for a pair of truly unrelated individuals.

There are multiple ways to partition the sample into an ancestry representative 'unrelated subset' and a 'related subset'. In all of the scenarios described below, the set of all samples is limited to those in sample.include when it is specified (i.e. not NULL):

If kinobj is specified, divobj is specified, and unrel.set = NULL, then the PC-AiR algorithm is used to find an 'optimal' partition of all samples (see 'References' for a paper describing the PC-AiR algorithm).

If kinobj is specified, divobj is specified, and unrel.set is specified, then all individuals with IDs in unrel.set are forced in the 'unrelated subset' and the PC-AiR algorithm is used to partition the rest of the sample; this is especially useful for including reference samples of known ancestry in the 'unrelated subset'.

If kinobj is specified, and divobj = NULL, then kinobj is used to define the unrelated set but ancestry divergence is ignored.

If kinobj = NULL, and unrel.set is specified, then all individuals with IDs in unrel.set are put in the 'unrelated subset' and the rest of the individuals are put in the 'related subset'.

If kinobj = NULL, and unrel.set = NULL, then the function will perform a "standard" PCA analysis.

NOTE: kinobj and divobj may be identical.

All pcair methods take the same arguments, as they ultimately call the gds.class method. The MatrixGenotypeReader method is implemented by writing a temporary GDS file.

Value

An object of class 'pcair'. A list including:

vectors	A matrix of principal components; each column is a principal component. Sample IDs are provided as rownames. The number of PCs returned can be adjusted by supplying the eigen.cnt argument, which is passed to snpgdsPCA.
values	A vector of eigenvalues matching the principal components. These values are determined from the standard PCA run on the 'unrelated subset'.
rels	A vector of IDs for individuals in the 'related subset'.
unrels	A vector of IDs for individuals in the 'unrelated subset'.
kin.thresh	The threshold value used for declaring each pair of individuals as related or unrelated.
div.thresh	The threshold value used for determining if each pair of individuals is ancestrally divergent.
sample.id	A vector of IDs for the samples used in the analysis.
nsamp	The total number of samples in the analysis.
nsnps	The total number of SNPs used in the analysis.

varprop	The variance proportion for each principal component.
call	The function call passed to pcair.
method	A character string. Either "PC-AiR" or "Standard PCA" identifying which method was used for computing principal components. "Standard PCA" is used if no relatives were identified in the sample.

Author(s)

Matthew P. Conomos

References

Conomos M.P., Miller M., & Thornton T. (2015). Robust Inference of Population Structure for Ancestry Prediction and Correction of Stratification in the Presence of Relatedness. Genetic Epidemiology, 39(4), 276-293.

Manichaikul, A., Mychaleckyj, J.C., Rich, S.S., Daly, K., Sale, M., & Chen, W.M. (2010). Robust relationship inference in genome-wide association studies. Bioinformatics, 26(22), 2867-2873.

See Also

pcairPartition for a description of the function used by pcair that can be used to partition the sample into 'unrelated' and 'related' subsets without performing PCA. plot.pcair for plotting. kingToMatrix for creating a matrix of pairwise kinship coefficient estimates from KING output text files that can be used for kinobj or divobj. GWASTools for a description of the package containing the following functions: GenotypeData for a description of creating a GenotypeData class object for storing sample and SNP genotype data, MatrixGenotypeReader for a description of reading in genotype data stored as a matrix, and GdsGenotypeReader for a description of reading in genotype data stored as a GDS file. Also see snpgdsBED2GDS in the SNPRelate package for a description of converting binary PLINK files to GDS. The generic functions summary and print.

Examples

```
# file path to GDS file
gdsfile <- system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS")
# read in GDS data
HapMap_geno <- gdsfmt::openfn.gds(gdsfile)
# load saved matrix of KING-robust estimates
data("HapMap_ASW_MXL_KINGmat")
# run PC-AiR
mypcair <- pcair(HapMap_geno, kinobj = HapMap_ASW_MXL_KINGmat,
                 divobj = HapMap_ASW_MXL_KINGmat)
gdsfmt::closefn.gds(HapMap_geno)
```

pcairPartition *Partition a sample into an ancestry representative 'unrelated subset' and a 'related subset'*

Description

pcairPartition is used to partition a sample from a genetic study into an ancestry representative 'unrelated subset' and a 'related subset'. The 'unrelated subset' contains individuals who are all mutually unrelated to each other and representative of the ancestries of all individuals in the sample, and the 'related subset' contains individuals who are related to someone in the 'unrealted subset'.

Usage

```
pcairPartition(kinobj, divobj = NULL,
               kin.thresh = 2^(-11/2), div.thresh = -2^(-11/2),
               unrel.set = NULL, sample.include = NULL, verbose = TRUE)
```

Arguments

kinobj	A symmetric matrix of pairwise kinship coefficients for every pair of individuals in the sample: upper and lower triangles must both be filled; diagonals should be self-kinship or set to a non-missing constant value. This matrix is used for partitioning the sample into the 'unrelated' and 'related' subsets. See 'Details' for how this interacts with kin.thresh and unrel.set. IDs for each individual must be set as the column names of the matrix. This matrix may also be provided as a GDS object; see 'Details'.
divobj	A symmetric matrix of pairwise ancestry divergence measures for every pair of individuals in the sample: upper and lower triangles must both be filled; diagonals should be set to a non-missing constant value. This matrix is used for partitioning the sample into the 'unrelated' and 'related' subsets. See 'Details' for how this interacts with div.thresh. IDs for each individual must be set as the column names of the matrix.This matrix may be identical to kinobj. This matrix may be NULL to ignore ancestry divergence. This matrix may also be provided as a GDS object; see 'Details'.
kin.thresh	Threshold value on kinobj used for declaring each pair of individuals as related or unrelated. The default value is 2^(-11/2) ~ 0.022, corresponding to 4th degree relatives. See 'Details' for how this interacts with kinobj.
div.thresh	Threshold value on divobj used for deciding if each pair of individuals is ancestrally divergent. The default value is -2^(-11/2) ~ -0.022. See 'Details' for how this interacts with divobj.
unrel.set	An optional vector of IDs for identifying individuals that are forced into the unrelated subset. See 'Details' for how this interacts with kinobj.
sample.include	An optional vector of IDs for selecting samples to consider for either set.
verbose	Logical indicator of whether updates from the function should be printed to the console; the default is TRUE.

Details

We recommend using software that accounts for population structure to estimate pairwise kinship coefficients to be used in `kinobj`. Any pair of individuals with a pairwise kinship greater than `kin.thresh` will be declared 'related.' Kinship coefficient estimates from the KING-robust software are typically used as measures of ancestry divergence in `divobj`. Any pair of individuals with a pairwise divergence measure less than `div.thresh` will be declared ancestrally 'divergent'. Typically, `kin.thresh` and `div.thresh` are set to be the amount of error around 0 expected in the estimate for a pair of truly unrelated individuals. If `unrel.set = NULL`, the PC-AiR algorithm is used to find an 'optimal' partition (see 'References' for a paper describing the algorithm). If `unrel.set` and `kinobj` are both specified, then all individuals with IDs in `unrel.set` are forced in the 'unrelated subset' and the PC-AiR algorithm is used to partition the rest of the sample; this is especially useful for including reference samples of known ancestry in the 'unrelated subset'.

For large sample sizes, storing both `kinobj` and `divobj` in memory may be prohibitive. Both matrices may be stored in GDS files and provided as `gds.class` objects. `mat2gds` saves matrices in GDS format. Alternatively, `kinobj` (but not `divobj`) can be represented as a sparse `Matrix` object; see `kingToMatrix` and `pcrelateToMatrix`.

Matrix objects from the **Matrix** package are also supported.

Value

A list including:

`rels`	A vector of IDs for individuals in the 'related subset'.
`unrels`	A vector of IDs for individuals in the 'unrelated subset'.

Note

`pcairPartition` is called internally in the function `pcair` but may also be used on its own to partition the sample into an ancestry representative 'unrelated' subset and a 'related' subset without performing PCA.

Author(s)

Matthew P. Conomos

References

Conomos M.P., Miller M., & Thornton T. (2015). Robust Inference of Population Structure for Ancestry Prediction and Correction of Stratification in the Presence of Relatedness. Genetic Epidemiology, 39(4), 276-293.

Manichaikul, A., Mychaleckyj, J.C., Rich, S.S., Daly, K., Sale, M., & Chen, W.M. (2010). Robust relationship inference in genome-wide association studies. Bioinformatics, 26(22), 2867-2873.

See Also

`pcair` which uses this function for finding principal components in the presence of related individuals. `kingToMatrix` for creating a matrix of kinship coefficent estimates or pairwise ancestry divergence measures from KING output text files that can be used as `kinobj` or `divobj`. `kin2gds` and `mat2gds` for saving kinship matrices to GDS.

Examples

```
# load saved matrix of KING-robust estimates
data("HapMap_ASW_MXL_KINGmat")
# partition the sample
part <- pcairPartition(kinobj = HapMap_ASW_MXL_KINGmat,
                       divobj = HapMap_ASW_MXL_KINGmat)
```

---

pcrelate *PC-Relate: Model-Free Estimation of Recent Genetic Relatedness*

Description

pcrelate is used to estimate kinship coefficients, IBD sharing probabilities, and inbreeding coefficients using genome-wide SNP data. PC-Relate accounts for population structure (ancestry) among sample individuals through the use of ancestry representative principal components (PCs) to provide accurate relatedness estimates due only to recent family (pedigree) structure.

Usage

```
## S4 method for signature 'GenotypeIterator'
pcrelate(gdsobj,
pcs,
scale = c('overall', 'variant', 'none'),
ibd.probs = TRUE,
sample.include = NULL,
training.set = NULL,
sample.block.size = 5000,
maf.thresh = 0.01,
maf.bound.method = c('filter', 'truncate'),
small.samp.correct = TRUE,
BPPARAM = bpparam(),
verbose = TRUE)
## S4 method for signature 'SeqVarIterator'
pcrelate(gdsobj,
pcs,
scale = c('overall', 'variant', 'none'),
ibd.probs = TRUE,
sample.include = NULL,
training.set = NULL,
sample.block.size = 5000,
maf.thresh = 0.01,
maf.bound.method = c('filter', 'truncate'),
small.samp.correct = TRUE,
BPPARAM = bpparam(),
verbose = TRUE)
samplesGdsOrder(gdsobj, sample.include)
calcISAFBeta(gdsobj,
```

```
        pcs,
        sample.include,
        training.set = NULL,
    BPPARAM = bpparam(),
        verbose = TRUE)
pcrelateSampBlock(gdsobj,
        betaobj,
        pcs,
        sample.include.block1,
        sample.include.block2,
scale = c('overall', 'variant', 'none'),
ibd.probs = TRUE,
maf.thresh = 0.01,
maf.bound.method = c('filter', 'truncate'),
BPPARAM = bpparam(),
verbose = TRUE)
correctKin(kinBtwn, kinSelf,
        pcs,
        sample.include = NULL)
correctK2(kinBtwn, kinSelf,
        pcs,
        sample.include = NULL,
        small.samp.correct = TRUE)
correctK0(kinBtwn)
```

Arguments

gdsobj	An object of class SeqVarIterator from the package **SeqVarTools**, or an object of class GenotypeIterator from the package **GWASTools**, containing the genotype data for the variants and samples to be used for the analysis.
pcs	A matrix of principal components (PCs) to be used for ancestry adjustment. Each column represents a PC, and each row represents an individual. IDs for each individual must be set as the row names of the matrix.
scale	A character string taking the values 'overall', 'variant', or 'none' indicating how genotype values should be standardized. This should be set to 'overall' (the default) in order to do a PC-Relate analysis; see 'Details' for more information.
ibd.probs	Logical indicator of whether pairwise IBD sharing probabilities (k0, k1, k2) should be estimated; the default is TRUE.
sample.include	A vector of IDs for samples to include in the analysis. If NULL, all samples in gdsobj are included.
training.set	An optional vector of IDs identifying which samples to use for estimation of the ancestry effect when estimating individual-specific allele frequencies. If NULL, all samples in sample.include are used. See 'Details' for more information.
sample.block.size	The number of individuals to read-in/analyze at once; the default value is 5000. See 'Details' for more information.

`maf.thresh` Minor allele frequency threshold; if an individual's estimated individual-specific minor allele frequency at a SNP is less than this value, that indivdiual will either have that SNP excluded from the analysis or have their estimated indivdiual-specific minor allele frequency truncated to this value, depending on `maf.bound.method`. The default value is 0.01.

`maf.bound.method` How individual-specific minor allele frequency estimates less that `maf.thresh` are handled. When set to 'filter' (default), SNPs for which an individual's estimated individual-specific minor allele frequency are below `maf.thresh` are excluded from the analysis for that individual. When set to 'truncate', estimated individual-specific minor allele frequncies below `maf.thresh` have their value set to `maf.thresh`.

`small.samp.correct` Logical indicator of whether to implement a small sample correction. The default is `TRUE`, but must be set to `FALSE` if `sample.block.size` is less than the number of samples or if `scale` is `'none'`.

`BPPARAM` A `BiocParallelParam` object to process blocks of variants in parallel. If not provided, the default back-end returned by `bpparam` will be used.

`verbose` Logical indicator of whether updates from the function should be printed to the console; the default is TRUE.

`betaobj` Outut of `calcISAFBeta`.

`sample.include.block1` A vector of IDs for samples to include in block 1.

`sample.include.block2` A vector of IDs for samples to include in block 2.

`kinBtwn` Output of `pcrelateSampBlock`.

`kinSelf` Output of `pcrelateSampBlock`.

Details

The basic premise of PC-Relate is to estimate kinship coefficients, IBD sharing probabilities, and inbreeding coefficients that reflect recent family (pedigree) relatedness by conditioning out genetic similarity due to distant population structure (ancestry) with ancestry representative principal components (PCs).

It is important that the PCs used in `pcs` to adjust for ancestry are representative of ancestry and NOT family structure, so we recommend using PCs calculated with PC-AiR (see: `pcair`).

`pcrelate` uses the `BiocParallel` package to process iterator chunks in parallel. See the `BiocParallel` documentation for more information on the default behaviour of `bpparam` and how to register different parallel backends. If serial execution is desired, set `BPPARAM=BiocParallel::SerialParam()`. Note that parallel execution requires more RAM than serial execution.

In order to perform relatedness estimation, allele frequency estimates are required for centering and scaling genotype values. Individual-specific allele frequencies calculated for each individual at each SNP using the PCs specified in `pcs` are used. There are muliple choices for how genotype values are scaled. When `scale` is 'variant', centered genotype values at each SNP are divided by their expected variance under Hardy-Weinberg equilibrium. When `scale` is 'overall', centered genotype values at all SNPs are divided by the average across all SNPs of their expected variances under

Hardy-Weinberg equilibrium; this scaling leads to more stable behavior when using low frequency variants. When `scale` is 'none', genotype values are only centered and not scaled; this won't provide accurate kinship coefficient estimates but may be useful for other purposes. Set `scale` to 'overall' to perform a standard PC-Relate analysis; this is the default. If `scale` is set to 'variant', the estimators are very similar to REAP.

The optional input `training.set` allows the user to specify which samples are used to estimate the ancestry effect when estimating individual-specific allele frequencies. Ideally, `training.set` is a set of mutually unrelated individuals. If prior information regarding pedigree structure is available, this can be used to select `training.set`, or if `pcair` was used to obtain the PCs, then the individuals in the PC-AiR 'unrelated subset' can be used. If no prior information is available, all individuals should be used.

The `sample.block.size` can be specified to alleviate memory issues when working with very large data sets. If `sample.block.size` is smaller than the number of individuals included in the analysis, then individuals will be analyzed in separate blocks. This reduces the memory required for the analysis, but genotype data must be read in multiple times for each block (to analyze all pairs), which increases the number of computations required.

`calcISAFBeta` and `pcrelateSampBlock` are provided as separate functions to allow parallelization for large sample sizes. `pcrelate` calls both of these functions internally. When calling these functions separately, use `samplesGdsOrder` to ensure the `sample.include` argument is in the same order as the GDS file. Use `correctKin`, `correctK2`, and `correctK0` after all sample blocks have been completed.

Value

An object of class '`pcrelate`'. A list including:

`kinBtwn`	A data.frame of estimated pairwise kinship coefficients and IBD sharing probabilities (if `ibd.probs` is `TRUE`).
`kinSelf`	A data.frame of estimated inbreeding coefficients.

Author(s)

Matthew P. Conomos

References

Conomos M.P., Reiner A.P., Weir B.S., & Thornton T.A. (2016). Model-free Estimation of Recent Genetic Relatedness. American Journal of Human Genetics, 98(1), 127-148.

See Also

`pcrelateToMatrix`

Examples

```
library(GWASTools)

# file path to GDS file
gdsfile <- system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS")
```

```
# read in GDS data
HapMap_geno <- GdsGenotypeReader(filename = gdsfile)
# create a GenotypeData class object
HapMap_genoData <- GenotypeData(HapMap_geno)
# load saved matrix of KING-robust estimates
data("HapMap_ASW_MXL_KINGmat")
# run PC-AiR
mypcair <- pcair(HapMap_genoData, kinobj = HapMap_ASW_MXL_KINGmat,
                 divobj = HapMap_ASW_MXL_KINGmat)

# create a GenotypeBlockIterator object
HapMap_genoData <- GenotypeBlockIterator(HapMap_genoData)
# run PC-Relate
mypcrel <- pcrelate(HapMap_genoData, pcs = mypcair$vectors[,1,drop=FALSE],
training.set = mypcair$unrels,
BPPARAM=BiocParallel::SerialParam())
head(mypcrel$kinBwtn)
head(mypcrel$kinSelf)

grm <- pcrelateToMatrix(mypcrel)
dim(grm)

close(HapMap_genoData)
```

`pcrelateToMatrix` *Creates a Genetic Relationship Matrix (GRM) of Pairwise Kinship Coefficient Estimates from PC-Relate Output*

Description

`pcrelateToMatrix` is used to create a genetic relationship matrix (GRM) of pairwise kinship coefficient estimates from the output of `pcrelate`.

Usage

```
## S4 method for signature 'pcrelate'
pcrelateToMatrix(pcrelobj, sample.include = NULL, thresh = NULL, scaleKin = 2,
                 verbose = TRUE)
```

Arguments

`pcrelobj`	The object containing the output from `pcrelate`. This should be a list of class `pcrelate` containing two data.frames; `kinSelf` with inbreeding coefficient estimates and `kinBtwn` with pairwise kinship coefficient estimates.
`sample.include`	A vector of IDs for samples to be included in the GRM. The default is NULL, which includes all samples in `pcrelobj`.
`thresh`	Kinship threshold for clustering samples to make the output matrix sparse block-diagonal. This thresholding is done after scaling kinship values by `scaleKin`. When `NULL`, no clustering is done. See 'Details'.

scaleKin	Specifies a numeric constant to scale each estimated kinship coefficient by in the GRM. The default value is 2.
verbose	Logical indicator of whether updates from the function should be printed to the console; the default is TRUE.

Details

This function provides a quick and easy way to construct a genetic relationship matrix (GRM) from the output of pcrelate.

thresh sets a threhsold for clustering samples such that any pair with an estimated kinship value greater than thresh is in the same cluster. All pairwise estimates within a cluster are kept, even if they are below thresh. All pairwise estimates between clusters are set to 0, creating a sparse, block-diagonal matrix. When thresh is NULL, no clustering is done and all samples are returned in one block. This feature may be useful for creating a sparse GRM when running association tests with very large sample sizes. Note that thresholding is done after scaling kinship values by scaleKin.

Value

An object of class 'Matrix' with pairwise kinship coefficients.

Author(s)

Matthew P. Conomos

References

Conomos M.P., Reiner A.P., Weir B.S., & Thornton T.A. (2016). Model-free Estimation of Recent Genetic Relatedness. American Journal of Human Genetics, 98(1), 127-148.

See Also

pcrelate for the function that performs PC-Relate.

---

plot.pcair *PC-AiR: Plotting PCs*

Description

plot.pcair is used to plot pairs of principal components contained in a class 'pcair' object obtained as output from the pcair function.

Usage

```
## S3 method for class 'pcair'
plot(x, vx = 1, vy = 2, pch = NULL, col = NULL,
        xlim = NULL, ylim = NULL, main = NULL, sub = NULL,
        xlab = NULL, ylab = NULL, ...)
```

Arguments

x	An object of class 'pcair' obtained as output from the pcair function.
vx	An integer indicating which principal component to plot on the x-axis; the default is 1.
vy	An integer indicating which principal component to plot on the y-axis; the default is 2.
pch	Either an integer specifying a symbol or a single character to be used in plotting points. If NULL, the default is dots for the 'unrelated subset' and + for the 'related subset'.
col	A specification for the plotting color for points. If NULL, the default is black for the 'unrelated subset' and blue for the 'related subset'.
xlim	The range of values shown on the x-axis. If NULL, the default shows all points.
ylim	The range of values shown on the y-axis. If NULL, the default shows all points.
main	An overall title for the plot. If NULL, the default specifies which PC-AiR PCs are plotted.
sub	A sub title for the plot. If NULL, the default is none.
xlab	A title for the x-axis. If NULL, the default specifies which PC-AiR PC is plotted.
ylab	A title for the y-axis. If NULL, the default specifies which PC-AiR PC is plotted.
...	Other parameters to be passsed through to plotting functions, (see par).

Details

This function provides a quick and easy way to plot principal components obtained with the function pcair to visualize the population structure captured by PC-AiR.

Value

A figure showing the selected principal components plotted against each other.

Author(s)

Matthew P. Conomos

See Also

pcair for obtaining principal components that capture population structure in the presence of relatedness. par for more in depth descriptions of plotting parameters. The generic function plot.

Examples

```
# file path to GDS file
gdsfile <- system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS")
# read in GDS data
HapMap_geno <- gdsfmt::openfn.gds(gdsfile)
# load saved matrix of KING-robust estimates
data("HapMap_ASW_MXL_KINGmat")
```

```
# run PC-AiR
mypcair <- pcair(HapMap_geno, kinobj = HapMap_ASW_MXL_KINGmat,
                 divobj = HapMap_ASW_MXL_KINGmat)
# plot top 2 PCs
plot(mypcair)
# plot PCs 3 and 4
plot(mypcair, vx = 3, vy = 4)
gdsfmt::closefn.gds(HapMap_geno)
```

`print.pcair` *PC-AiR: Principal Components Analysis in Related Samples*

Description

Print methods for pcair

Usage

```
## S3 method for class 'pcair'
print(x, ...)
## S3 method for class 'pcair'
summary(object, ...)
## S3 method for class 'summary.pcair'
print(x, ...)
```

Arguments

`object`	An object of class 'pcair', i.e. output from the `pcair` function.
`x`	An object of class 'pcair', i.e. output from the `pcair` function.
`...`	Further arguments passed to or from other methods.

Author(s)

Matthew P. Conomos

See Also

`pcair` for obtaining principal components that capture population structure in the presence of relatedness.

Examples

```
# file path to GDS file
gdsfile <- system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS")
# read in GDS data
HapMap_geno <- gdsfmt::openfn.gds(gdsfile)
# load saved matrix of KING-robust estimates
data("HapMap_ASW_MXL_KINGmat")
```

```
# run PC-AiR
mypcair <- pcair(HapMap_geno, kinobj = HapMap_ASW_MXL_KINGmat,
                 divobj = HapMap_ASW_MXL_KINGmat)
print(mypcair)
summary(mypcair)
gdsfmt::closefn.gds(HapMap_geno)
```

---

sample_annotation_1KG *Annotation for 1000 genomes Phase 3 samples*

Description

Annotation for 1000 genomes Phase 3 samples included in the VCF files in "extdata/1KG".

Usage

```
data(sample_annotation_1KG)
```

Format

A data.frame with columns:

- sample.idSample identifier
- PopulationPopulation of sample
- sexSex of sample

Source

ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp

References

A global reference for human genetic variation, The 1000 Genomes Project Consortium, Nature 526, 68-74 (01 October 2015) doi:10.1038/nature15393.

---

varCompCI *Variance Component Confidence Intervals*

Description

`varCompCI` provides confidence intervals for the variance component estimates found using `fitNullModel`. The confidence intervals can be found on either the original scale or for the proportion of total variability explained.

Usage

```
varCompCI(null.model, prop = TRUE)
```

Arguments

null.model	A null model object returned by fitNullModel.
prop	A logical indicator of whether the point estimates and confidence intervals should be returned as the proportion of total variability explained (TRUE) or on the orginal scale (FALSE).

Details

varCompCI takes the object returned by fitNullModel as its input and returns point estimates and confidence intervals for each of the random effects variance component estimates. If a kinship matrix or genetic relationship matrix (GRM) was included as a random effect in the model fit using fitNullModel, then this function can be used to provide a heritability estimate when prop is TRUE.

Value

varCompCI prints a table of point estimates and 95% confidence interval limits for each estimated variance component.

Author(s)

Matthew P. Conomos

See Also

fitNullModel for fitting the mixed model and performing the variance component estimation.

Examples

```
library(GWASTools)

# file path to GDS file
gdsfile <- system.file("extdata", "HapMap_ASW_MXL_geno.gds", package="GENESIS")
# read in GDS data
HapMap_geno <- GdsGenotypeReader(filename = gdsfile)
# create a GenotypeData class object
HapMap_genoData <- GenotypeData(HapMap_geno)
# load saved matrix of KING-robust estimates
data("HapMap_ASW_MXL_KINGmat")

# run PC-AiR
mypcair <- pcair(HapMap_genoData, kinobj = HapMap_ASW_MXL_KINGmat,
                 divobj = HapMap_ASW_MXL_KINGmat)

# run PC-Relate
HapMap_genoData <- GenotypeBlockIterator(HapMap_genoData, snpBlock=20000)
mypcrel <- pcrelate(HapMap_genoData, pcs = mypcair$vectors[,1,drop=FALSE],
     training.set = mypcair$unrels,
                      BPPARAM = BiocParallel::SerialParam())
close(HapMap_genoData)

# generate a phenotype
```

```
set.seed(4)
pheno <- 0.2*mypcair$vectors[,1] + rnorm(mypcair$nsamp, mean = 0, sd = 1)

annot <- data.frame(sample.id = mypcair$sample.id,
                    pc1 = mypcair$vectors[,1], pheno = pheno)

# make covariance matrix
cov.mat <- pcrelateToMatrix(mypcrel, verbose=FALSE)[annot$sample.id, annot$sample.id]

# fit the null mixed model
nullmod <- fitNullModel(annot, outcome = "pheno", covars = "pc1", cov.mat = cov.mat)

# find the variance component CIs
varCompCI(nullmod, prop = TRUE)
varCompCI(nullmod, prop = FALSE)
```

Index