Package: podkat (via r-universe)

November 1, 2026

Type Package

Title Position-Dependent Kernel Association Test

Version 1.45.0

Date 2025-09-19

Maintainer Ulrich Bodenhofer <ulrich@bodenhofer.com>

Description This package provides an association test that is capable of dealing with very rare and even private variants. This is accomplished by a kernel-based approach that takes the positions of the variants into account. The test can be used for pre-processed matrix data, but also directly for variant data stored in VCF files. Association testing can be performed whole-genome, whole-exome, or restricted to pre-defined regions of interest. The test is complemented by tools for analyzing and visualizing the results.

URL https://github.com/UBod/podkat

License GPL (>= 2)

Depends R (>= 3.5.0), methods, Rsamtools (>= 1.99.1), GenomicRanges

Imports Rcpp (>= 0.11.1), parallel, stats (>= 4.3.0), graphics, grDevices, utils, Biobase, BiocGenerics, Matrix, Seqinfo, IRanges, Biostrings, BSgenome (>= 1.32.0)

Suggests BSgenome.Hsapiens.UCSC.hg38.masked, TxDb.Hsapiens.UCSC.hg38.knownGene, BSgenome.Mmusculus.UCSC.mm10.masked, GWASTools (>= 1.13.24), VariantAnnotation, SummarizedExperiment, knitr

LinkingTo Rcpp, Rhtslib (>= 1.15.3)

SystemRequirements GNU make

VignetteBuilder knitr

Collate AllGenerics.R AllClasses.R inputChecks.R sort-methods.R show-methods.R print-methods.R summary-methods.R p.adjust-methods.R c-methods.R access-methods.R coerce-methods.R resampling.R unmaskedRegions.R

partitionRegions-methods.R genotypeMatrix-methods.R computeKernel.R computePvalues.R readGenotypeMatrix-methods.R readVariantInfo-methods.R readSampleNamesFromVcfHeader.R readRegionsFromBedFile.R weightFuncs.R assocTest-methods.R nullModel-methods.R qqplot-methods.R plot-methods.R filterResult-methods.R split-methods.R computeWeights.R weights-methods.R

biocViews Genetics, WholeGenome, Annotation, VariantAnnotation, Sequencing, DataImport

NeedsCompilation yes

Config/pak/sysreqs make libbz2-dev liblzma-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-05-13 13:05:42 UTC

RemoteUrl https://github.com/bioc/podkat

RemoteRef HEAD

RemoteSha a5ee2c46535d6071db4ae2927c3d98390238b392

Contents

podkat-package . . . 3
assocTest . . . 4
AssocTestResult-class . . . 10
AssocTestResultRanges-class . . . 11
computeKernel . . . 14
filterResult-methods . . . 16
GenotypeMatrix-class . . . 18
genotypeMatrix-methods . . . 20
hgA . . . 24
nullModel . . . 25
NullModel-class . . . 29
p.adjust-methods . . . 32
partitionRegions-methods . . . 33
plot . . . 35
print-methods . . . 39
qqplot . . . 41
readGenotypeMatrix-methods . . . 43
readRegionsFromBedFile . . . 46
readSampleNamesFromVcfHeader . . . 47
readVariantInfo-methods . . . 48
sort-methods . . . 51
split-methods . . . 53
unmasked-datasets . . . 54
unmaskedRegions . . . 56
VariantInfo-class . . . 58
weightFuncs . . . 59
weights . . . 61

Index **64**

---

podkat-package *PODKAT Package*

Description

This package provides an association test that is capable of dealing with very rare and even private variants. This is accomplished by a kernel-based approach that takes the positions of the variants into account. The test can be used for pre-processed matrix data, but also directly for variant data stored in VCF files. Association testing can be performed whole-genome, whole-exome, or restricted to pre-defined regions of interest. The test is complemented by tools for analyzing and visualizing the results.

Details

The central method of this package is `assocTest`. It provides several different kernel-based association tests, in particular, the position-dependent kernel association test (PODKAT), but also some variants of the SNP-set kernel association test (SKAT). The test can be run for genotype data given in (sparse) matrix format as well as directly on genotype data stored in a variant call format (VCF) file. In any case, the user has to create a null model by the `nullModel` function beforehand. Upon completion of an association test, the package also provides methods for filtering, sorting, multiple testing correction, and visualization of results.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.lin <- nullModel(y ~ ., pheno)
```

```
## perform association test
res <- assocTest(Z, nm.lin, windows)

## display results
print(res)
print(p.adjust(res))
plot(p.adjust(res), which="p.value.adj")
```

---

assocTest *Perform Association Test*

Description

Method for performing a kernel-based association test given a genotype, VCF file, or kernel matrix

Usage

```
## S4 method for signature 'GenotypeMatrix,NullModel'
assocTest(Z, model, ranges,
          kernel=c("linear.podkat", "localsim.podkat",
                   "quadratic.podkat", "linear.SKAT",
                   "localsim.SKAT", "quadratic.SKAT"),
          width=1000, weights=NULL, weightFunc=betaWeights(),
          method=NULL, adj=c("automatic", "none", "force"),
          pValueLimit=0.05)
## S4 method for signature 'matrix,NullModel'
assocTest(Z, model, method=NULL,
          adj=c("automatic", "none", "force"), pValueLimit=0.05)
## S4 method for signature 'TabixFile,NullModel'
assocTest(Z, model, ranges,
          kernel=c("linear.podkat", "localsim.podkat",
                   "quadratic.podkat", "linear.SKAT",
                   "localsim.SKAT", "quadratic.SKAT"),
          cl=NULL, nnodes=1, batchSize=20,
          noIndels=TRUE, onlyPass=TRUE, na.limit=1, MAF.limit=1,
          na.action=c("impute.major", "omit"),
          MAF.action=c("invert", "omit", "ignore"),
          sex=NULL, weightFunc=betaWeights(), width=1000,
          method=NULL, adj=c("automatic", "none", "force"),
          pValueLimit=(0.1 / length(ranges)), tmpdir=tempdir(),
          displayProgress=TRUE)
## S4 method for signature 'character,NullModel'
assocTest(Z, model, ...)
```

Arguments

Z	an object of class GenotypeMatrix, a quadratic kernel matrix, an object of class TabixFile, or a character string with a file name
model	an object of class NullModel
ranges	an object with genomic regions to be tested; may be an object of class GRanges or GRangesList. If missing, assocTest takes the whole genotype matrix or the genotypes in the VCF file as a whole.
kernel	determines the kernel that should be used for association testing (see Subsection 9.2 of the package vignette for details)
width	tolerance radius parameter for position-dependent kernels "linear.podkat", "quadratic.podkat", and "localsim.podkat"; must be single positive numeric value; ignored for kernels "linear.SKAT", "quadratic.SKAT", and "localsim.SKAT" (see Subsection 9.2 of the package vignette for details)
weights	for the method with signature GenotypeMatrix,NullModel, it is also possible to supply weights directly as a numeric vector that is as long as the number of columns of Z. In this case, the argument weightFunc is ignored. Use NULL (default) to use automatic weighting with the function supplied as argument weightFunc. If weightFunc is NULL too, no weighting takes place, i.e. an unweighted kernel is used.
weightFunc	function for computing variant weights from minor allele frequencies (MAFs); see weightFuncs for weighting and Subsection 9.3 of the package vignette for functions provided by the **podkat** package. Use NULL for unweighted kernels.
method	identifies the method for computing the p-values. If the null model is of type "logistic" and small sample correction is applied (see argument adj below), possible values are "unbiased", "population", "sample", and "SKAT" (see details below and Subsection 9.5 of the package vignette). If the null model is of type "linear" or if the null model is of type "logistic" and no small sample correction is applied, possible values are "davies", "liu", and "liu.mod" (see details below and Subsection 9.1 of the package vignette). If the null model is of type "bernoulli", this argument is ignored.
adj	whether or not to use small sample correction for logistic models (binary trait with covariates). The choice "none" turns off small sample correction. If "force" is chosen, small sample correction is turned on unconditionally. If "automatic" is chosen (default), small sample correction is turned on if the number of samples does not exceed 2,000. This argument is ignored for any type of model except "logistic" and small sample correction is switched off. For details how to train a null model for small sample correction, see nullModel and Sections 4 and 9.5 of the package vignette. An adjustment of higher moments is performed whenever sampled null model residuals are available in the null model model (slot res.resampled.adj, see NullModel).
pValueLimit	if the null model is of type "bernoulli", assocTest performs an exact mixture of Bernoulli test. This test uses a combinatorial algorithm to compute exact p-values and, for the sake of computational efficiency, quits if a prespecified p-value threshold is exceeded. This threshold can be specified with the pValueLimit argument. This argument is ignored for other types of tests/null models.

`cl` if `cl` is an object of class `SOCKcluster`, association testing is carried out in parallel on the cluster specified by `cl`. If `NULL` (default), either no parallelization is done (if `nnodes=1`) or `assocTest` launches a cluster with `nnodes` R client processes on `localhost`. See Subsection 8.5.2 of the package vignette.

`nnodes` if `cl` is `NULL` and `nnodes` is greater than 1, `makePSOCKcluster` is called with `nnodes` nodes on `localhost`, i.e. `nnodes` R slave processes are launched on which association testing is carried out in parallel. The default is `1`. See Subsection 8.5.2 of the package vignette.

`batchSize` parameter which determines how many regions of `ranges` are processed at once. The larger `batchSize`, the larger the the batches that are read from the VCF file `Z`. A larger `batchSize` reduces the number of individual read operations, which improves performance. However, a larger `batchSize` also requires larger amounts of memory. A good choice of `batchSize`, therefore, depends on the size and sparseness of the VCF file and as well on the available memory. See Subsection 8.5 of the package vignette.

`noIndels` if `TRUE` (default), only single nucleotide variants (SNVs) are considered and indels in the VCF file `Z` are skipped.

`onlyPass` if `TRUE` (default), only variants are considered whose value in the `FILTER` column is "PASS".

`na.limit` all variants with a missing value ratio above this threshold in the VCF file `Z` are not considered.

`MAF.limit` all variants with a minor allele frequency (MAF) above this threshold in the VCF file `Z` are not considered.

`na.action` if "impute.major", all missing values will be imputed by major alleles before association testing. If "omit", all columns containing missing values in the VCF file `Z` are ignored.

`MAF.action` if "invert", all columns with an MAF exceeding 0.5 will be inverted in the sense that all minor alleles will be replaced by major alleles and vice versa. If "omit", all variants in the VCF file with an MAF greater than 0.5 are ignored. If "ignore", no action is taken and MAFs greater than 0.5 are kept as they are.

`sex` if `NULL`, all samples are treated the same without any modifications; if `sex` is a factor with levels `F` (female) and `M` (male) that is as long as `length(model)`, this argument is interpreted as the sex of the samples. In this case, the genotypes corresponding to male samples are doubled before further processing. This is designed for mixed-sex analyses of the X chromosome outside of the pseudoautosomal regions.

`tmpdir` if computations are parallelized over multiple client processes (see arguments `nnodes` and `cl`), the exchange of the null model object between the master process and the client processes is done via a temporary file. The `tmpdir` argument allows to specify into which directory the temporary file should be saved. On multi-core systems, the default should be sufficient. If the computations are distributed over a custom cluster, the `tmpdir` argument needs to be chosen such that all clients can access it via the same path.

`displayProgress` if `TRUE` (default) and if `ranges` is a `GRangesList`, a progress message is printed upon completion of each list component (typically consisting of regions of one chromosome); this argument is ignored if `ranges` is not an object of class `GRangesList`.

... all other parameters are passed on to the assocTest method with signature TabixFile,NullModel.

Details

The assocTest method is the main function of the **podkat** package. For a given genotype and a null model, it performs the actual association test(s).

For null models of types "linear" and "logistic" (see NullModel and nullModel), a *variance component score test* is used (see Subsection 9.1 of the package vignette for details). The test relies on the choice of a particular kernel to measure the pairwise similarities of genotypes. The choice of the kernel can be made with the kernel argument (see computeKernel and Subsection 9.2 of the package vignette for more details). For null models of type "linear", the test statistic follows a *mixture of chi-squares distribution*. For models of typ "logistic", the test statistic approximately follows a mixture of chi-squares distribution. The computation of p-values for a given mixture of chi-squares can be done according to *Davies (1980)* (which is the default), according to *Liu et al. (2009)*, or using a modified method similar to the one suggested by *Liu et al. (2009)* as implemented in the **SKAT** package, too. Which method is used can be controlled using the method argument. If method according to *Davies (1980)* fails, assocTest resorts to the method by *Liu et al. (2009)*. See also Subsection 9.1 of the package vignette for more details.

For null models of type "logistic", the assocTest method also offers the small sample correction suggested by *Lee et al. (2012)*. Whether small sample correction is applied, is controlled by the adj argument. The additional adjustment of higher moments as suggested by *Lee et al. (2012)* is performed whenever resampled null model residuals are available in the null model model (slot res.resampled.adj, see NullModel). In this case, the method argument controls how the excess kurtosis of test statistics sampled from the null distribution are computed. The default setting "unbiased" computes unbiased estimates by using the exact expected value determined from the mixture components. The settings "population" and "sample" use almost unbiased and biased sample statistics, respectively. The choice "SKAT" uses the same method as implemented in the **SKAT** package. See Subsection 9.5 of the package vignette for more details.

If the null model is of type "bernoulli", the test statistic follows a mixture of Bernoulli distributions. In this case, an exact p-value is determined that is computed as the probability to observe a test statistic for random Bernoulli-distributed traits (under the null hypothesis) that is at least as large as the observed test statistic. For reasons of computational complexity, this option is limited to sample numbers not larger than 100. See Subsection 9.1 of the package vignette for more details.

The **podkat** package offers multiple interfaces for association testing all of which require the second argument model to be a NullModel object. The simplest method is to call assocTest for an object of class GenotypeMatrix as first argument Z. If the ranges argument is not supplied, a single association test is performed using the entire genotype matrix contained in Z and an object of class AssocTestResult is returned. In this case, all variants need to reside on the same chromosome (compare with computeKernel). If the ranges argument is specified, each region in ranges is tested separately and the result is returned as an AssocTestResultRanges object.

As said, the simplest method is to store the entire genotype in a GenotypeMatrix object and to call assocTest as described above. This approach has the shortcoming that the entire genotype must be read (e.g. from a VCF file) and kept in memory as a whole. For large studies, in particular, whole-genome studies, this is not feasible. In order to be able to cope with large studies, the **podkat** package offers an interface that allows for reading from a VCF file piece by piece without the need to read and store the entire genotype at once. If Z is a TabixFile object or the name of a VCF file,

assocTest reads from the file in batches of batchSize regions, performs the association tests for these regions, and returns the results as an AssocTestResultRanges object. This sequential batch processing can also be parallelized. The user can either set up a cluster him-/herself and pass the SOCKcluster object as cl argument. If the cl is NULL, users can leave the setup of the cluster to assocTest. In this case, the only thing necessary is to determine the number of R client processes by the nnodes argument. The variant with the VCF interface supports the same pre-processing and filter arguments as readGenotypeMatrix to control which variants are actually taken into account and how to handle variants with MAFs greater than 50%.

If the argument Z is a numeric matrix, Z is interpreted as a kernel matrix K. Then a single association test is performed as described above and the result is returned as an AssocTestResult object. This allows the user to use a custom kernel not currently implemented in the **podkat** package. The assocTest function assumes that row and column objects in the kernel matrix are in the same order. It does not perform any check whether row and column names are the same or whether the kernel matrix is actually positive semi-definite. Users should be aware that running the function for invalid kernels matrices, i.e. for a matrix that is not positive semi-definite, produces meaningless results and may even lead to unexpected errors.

Finally, note that the samples in the null model model and in the genotype (GenotypeMatrix object or VCF file) need not be aligned to each other. If both the samples in model and in the genotype are named (i.e. row names are defined for Z if it is a GenotypeMatrix object; VCF files always contain sample names anyway), assocTest checks if all samples in model are present in the genotype. If so, it selects only those samples from the genotype that occur in the null model. If not, it quits with an error. If either the samples in the null model or the genotypes are not named, assocTest assumes that the samples are aligned to each other. This applies only if the number of samples in the null model and the number of genotypes are the same or if the number of genotypes equals the number of samples in the null model plus the number of samples that were omitted from the null model when it was trained (see NullModel and nullModel). Otherwise, the function quits with an error. An analogous procedure is applied if the kernel matrix interface is used (signature matrix,NullModel).

Value

an object of class AssocTestResult or AssocTestResultRanges (see details above)

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

Wu, M. C., Lee, S., Cai, T., Li, Y., Boehnke, M., and Lin, X. (2011) Rare-variant association testing for sequencing data with the sequence kernel association test. *Am. J. Hum. Genet.* **89**, 82-93. DOI: doi:10.1016/j.ajhg.2011.05.029.

Lee, S., Emond, M. J., Bamshad, M. J., Barnes, K. C., Rieder, M. J., Nickerson, D. A., NHLBI Exome Sequencing Project - ESP Lung Project Team, Christiani, D. C., Wurfel, M. M., and Lin, X. (2012) Optimal unified approach for rare-variant association testing with application to small-sample case-control whole-exome sequencing studies. *Am. J. Hum. Genet.* **91**, 224-237. DOI: doi:10.1016/j.ajhg.2012.06.007.

Davies, R. B. (1980) The distribution of a linear combination of χ^2 random variables. *J. R. Stat. Soc. Ser. C-Appl. Stat.* **29**, 323-333.

Liu, H., Tang, Y., and Zhang, H. (2009) A new chi-square approximation to the distribution of non-negative definite quadratic forms in non-central normal variables. *Comput. Stat. Data Anal.* **53**, 853-856.

See Also

AssocTestResult, AssocTestResultRanges, nullModel, NullModel, computeKernel, weightFuncs, readGenotypeMatrix, GenotypeMatrix, plot, qqplot, p.adjust, filterResult

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno.c <- read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
model.c <- nullModel(y ~ ., pheno.c)

## perform association test
res <- assocTest(Z, model.c, windows)
print(res)

## perform association test using the VCF interface
res <- assocTest(vcfFile, model.c, windows, batchSize=100)
print(res)

## create Manhattan plot of adjusted p-values
plot(p.adjust(res), which="p.value.adj")

## read phenotype data from CSV file (binary trait + covariates)
phenoFile <- system.file("examples/example1log.csv", package="podkat")
pheno.b <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
model.b <- nullModel(y ~ ., pheno.b)

## perform association test
res <- assocTest(Z, model.b, windows)
print(res)
```

```
## create Manhattan plot of adjusted p-values
plot(p.adjust(res), which="p.value.adj")
```

AssocTestResult-class *Class* AssocTestResult

Description

S4 class for storing the result of an association test for a single genomic region

Objects

Objects of this class are created by calling assocTest for a single genomic region.

Slots

The following slots are defined for AssocTestResult objects:

type: type of null model on which the association test was based

samples: character vector with sample names (if available, otherwise empty)

kernel: kernel that was used for the association test

dim: dimensions of genotype matrix that was tested

weights: weight vector that was used; empty if no weighting was performed

width: tolerance radius parameter that was used for position-dependent kernels

method: method(s) used to compute p-values; a single character string if no resampling was done, otherwise a list with two components specifying the p-value computation method for the test's p-value and the resampled p-values separately.

correction: a logical vector indicating whether the small sample correction was carried out (first component exact is TRUE) and/or higher moment correction was carried out (second component resampling is TRUE).

Q: test statistic

p.value: the test's p-value

Q.resampling: test statistics for sampled null model residuals

p.value.resampling: p-values for sampled null model residuals

p.value.resampled: estimated p-value computed as the relative frequency of p-values of sampled residuals that are at least as significant as the test's p-value

call: the matched call with which the object was created

Methods

show signature(object="AssocTestResult"): displays the test statistic and the p-value along with the type of the null model, the number of samples, the number of SNVs, and the kernel that was used to carry out the test.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

assocTest

Examples

```
## load genome description
data(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.lin <- nullModel(y ~ ., pheno)

## perform association test for entire genotype matrix
res <- assocTest(Z, nm.lin)
show(res)

## perform association test for subset of genotype matrix
res <- assocTest(Z[, 50:100], nm.lin)
show(res)
```

AssocTestResultRanges-class
Class AssocTestResultRanges

Description

S4 class for storing the result of an association test performed on multiple genomic regions

Objects

Objects of this class are created by calling assocTest with a non-empty ranges argument.

Slots

This class extends the class GRanges directly and therefore inherits all its slots and methods. The following slots are defined for AssocTestResultRanges objects additionally:

type: type of null model on which the association test was based

samples: character vector with sample names (if available, otherwise empty)

kernel: kernel that was used for the association test

weights: weight vector or weighting function that was used; NULL if no weighting was performed

width: tolerance radius parameter that was used for position-dependent kernels

adj.method: which method for multiple testing correction has been applied (if any)

vcfParams: list of parameters that were used for reading genotypes from VCF file

sex: factor with sex information (if any)

call: the matched call with which the object was created

Apart from these additional slots, all AssocTestResultRanges objects have particular metadata columns (accessible via mcols or elementMetadata):

n: number of variants tested in each region; a zero does not necessarily mean that there were no variants in this region, it only means that no variants were used for testing. Variants are omitted from the test if they do not show any variation or if they do not satisfy other filter criteria applied by assocTest. This metadata column is always present.

Q: test statistic for each region that was tested. This metadata column is always present.

p.value: p-value of test for each region that was tested. This metadata column is always present.

p.value.adj: adjusted p-value of test for each region that was tested. This metadata column is only present if multiple testing correction has been applied (see p.adjust).

p.value.resampled: estimated p-value computed as the relative frequency of p-values of sampled residuals that are at least as significant as the test's p-value in each region. This metadata column is only present if resampling has been applied, i.e. if assocTest has been called with n.resampling greater than zero.

p.value.resampled.adj: adjusted empirical p-value (see above). This metadata column is only present if resampling and multiple testing correction has been applied.

Methods

c signature(object="AssocTestResultRanges"): allows for concatenating two or more AssocTestResultRanges objects; this is only meaningful if the different tests have been performed on the same samples, on the same genome, with the same kernel, and with the same VCF reading parameters (in case that the association test has been performed directly on a VCF file). All these conditions are checked and if any of them is not fulfilled, the method quits with an error. Merging association test results that were computed with different sex parameters is possible, but the sex component is omitted and a warning is issued. Note that multiple testing correction (see p.adjust) should not be carried out on parts, but only on the entire set of all tests. That is why c strips off all adjusted p-values.

p.adjust signature(object="AssocTestResultRanges"): multiple testing correction, see p.adjust.

filterResult `signature(object="AssocTestResultRanges")`: apply filtering to p-values or adjusted p-values. For more details, see `filterResult`.

sort `signature(object="AssocTestResultRanges")`: sort `AssocTestResultRanges` object according to specified sorting criterion. See `sort` for more details.

plot `signature(object="AssocTestResultRanges")`: make a Manhattan plot of the association test result. See `plot` for more details.

qqplot `signature(object="AssocTestResultRanges")`: make quantile-quantile (Q-Q) plot of association test result. See `qqplot` for more details.

show `signature(object="AssocTestResultRanges")`: displays some general information about the result of the association test, such as, the number of samples, the number of regions tested, the number of regions without variants, the average number of variants in the tested regions, the genome, the kernel that was applied, and the type of multiple testing correction (if any).

print `signature(x="AssocTestResultRanges")`: allows for displaying more information about the object than `show`. See `print` for more details.

Accessors and subsetting

As mentioned above, the `AssocTestResultRanges` inherits all methods from the `GRanges` class.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

`assocTest`

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.lin <- nullModel(y ~ ., pheno)
```

```
## perform association test for multiple regions
res <- assocTest(Z, nm.lin, windows)

## perform multiple testing correction
res.adj <- p.adjust(res)
print(res.adj)

## show sorted results
as(sort(res.adj), "GRanges")

## show filtered result
print(filterResult(res.adj, cutoff=0.05, filterBy="p.value.adj"))

## make a Manhattan plot
plot(res.adj, which="p.value.adj")
```

`computeKernel` *Compute Kernel Matrix*

Description

Computes kernel matrix for a given genotype matrix

Usage

```
computeKernel(Z, kernel=c("linear.podkat", "quadratic.podkat",
              "localsim.podkat", "linear.SKAT", "quadratic.SKAT",
              "localsim.SKAT"), weights=NULL, pos=NULL, width=1000)
```

Arguments

`Z`	a matrix or an object of class `Matrix` (note that the latter also includes objects of class `GenotypeMatrix`)
`kernel`	type of kernel to use
`weights`	numeric vector with variant weights; must be as long as the number of columns of `Z`. Use `NULL` for unweighted kernels.
`pos`	numeric vector with positions of variants; must be as long as the number of columns of `Z`. This argument is mandatory for the position-dependent kernels "linear.podkat", "quadratic.podkat", and "localsim.podkat"; ignored for kernels "linear.SKAT", "quadratic.SKAT", and "localsim.SKAT".
`width`	tolerance radius parameter for position-dependent kernels "linear.podkat", "quadratic.podkat", and "localsim.podkat" (see details below); must be single positive numeric value. Ignored for kernels "linear.SKAT", "quadratic.SKAT", and "localsim.SKAT".

Details

This function computes a kernel matrix for a given genotype matrix `Z` and a given kernel. It supposes that `Z` is a matrix-like object (a numeric matrix, a sparse matrix, or an object of class `GenotypeMatrix`) in which rows correspond to samples and columns correspond to variants. There are six different kernels available: "linear.podkat", "quadratic.podkat", "localsim.podkat", "linear.SKAT", "quadratic.SKAT", and "localsim.SKAT". All of these kernels can be used with or without weights. The weights can be specified with the `weights` argument which must be a numeric vector with as many elements as the matrix `Z` has columns. If no weighting should be used, `weights` must be set to `NULL`.

The position-dependent kernels "linear.podkat", "quadratic.podkat", and "localsim.podkat" require the positions of the variants in `Z`. So, if any of these three kernels is selected, the argument `pos` is mandatory and must be a numeric vector with as many elements as the matrix `Z` has columns.

If the `pos` argument is `NULL` and `Z` is a `GenotypeMatrix` object, the positions in `variantInfo(Z)` are taken. In this case, all variants need to reside on the same chromosome. If the variants in `variantInfo(Z)` are from multiple chromosomes, `computeKernel` quits with an error. As said, this only happens if `pos` is `NULL`, otherwise the `pos` argument has priority over the information stored in `variantInfo(Z)`.

For details on how the kernels compute the pairwise similarities of genotypes, see Subsection 9.2 of the package vignette.

Value

a positive semi-definite kernel matrix with as many rows and columns as `Z` has rows

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

Wu, M. C., Lee, S., Cai, T., Li, Y., Boehnke, M., and Lin, X. (2011) Rare-variant association testing for sequencing data with the sequence kernel association test. *Am. J. Hum. Genet.* **89**, 82-93. DOI: doi:10.1016/j.ajhg.2011.05.029.

See Also

`GenotypeMatrix`

Examples

```
## create a toy example
A <- matrix(rbinom(50, 2, prob=0.2), 5, 10)
pos <- sort(sample(1:10000, ncol(A)))

## compute some unweighted kernels
computeKernel(A, kernel="linear.podkat", pos=pos, width=100)
computeKernel(A, kernel="localsim.podkat", pos=pos, width=100)
```

```
computeKernel(A, kernel="linear.SKAT")

## compute some weighted kernels
MAF <- colSums(A) / (2 * nrow(A))
weights <- betaWeights(MAF)
computeKernel(A, kernel="linear.podkat", pos=pos, weights=weights)
computeKernel(A, kernel="linear.SKAT", weights=weights)
computeKernel(A, kernel="localsim.SKAT", weights=weights)
```

`filterResult-methods` *Filter Association Test Results According to p-Values or Variants' Contributions*

Description

Given an `AssocTestResultRanges` object, this method filters regions according to p-values or variants' contributions.

Usage

```
## S4 method for signature 'AssocTestResultRanges'
filterResult(object, cutoff=0.05,
             filterBy=c("p.value", "p.value.adj", "p.value.resampled",
             "p.value.resampled.adj"))
## S4 method for signature 'GRanges'
filterResult(object, cutoff=0.1)
## S4 method for signature 'GRangesList'
filterResult(object, cutoff=0.1)
```

Arguments

`object`	object of class `AssocTestResultRanges`, `GRanges`, or `GRangesList`
`cutoff`	threshold
`filterBy`	according to which p-value column filtering should be done; the default is "p.value".

Details

If called for an `AssocTestResultRanges` object as first argument, this method strips off all regions the p-values of which exceed the threshold `cutoff`. By default, this filtering is applied to raw p-values (metadata column "p.value"). The `filterBy` argument allows for performing filtering on any of the other three p-value metadata columns (if available, otherwise the method quits with an error).

If called for a `GRanges` object as first argument, this method checks if the first argument `object` has a metadata column named "weight.contribution". If it exists, it returns a `GRanges` object with the elements of `object` that have a value greater than `cutoff` in the "weight.contribution" metadata column. If this metadata column does not exist, the method quits with an error.

If called for a GRangesList object as first argument object, this method applies the filterResult method for each of its list components and returns a GRangesList object. If any of the components of object does not have a metadata column named "weight.contribution", the method quits with an error.

Value

an object of class AssocTestResultRanges, GRanges, or GRangesList (see details above)

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

AssocTestResultRanges, p.adjust

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.lin <- nullModel(y ~ ., pheno)

## perform association test for multiple regions
res <- assocTest(Z, nm.lin, windows)
res.adj <- p.adjust(res, method="BH")

## show filtered results
res.f <- filterResult(res.adj)
print(res.f)
res.f <- filterResult(res.adj, filterBy="p.value.adj")
print(res.f)

## compute contributions
contrib <- weights(res.f, Z, nm.lin)
```

```
contrib

## extract most indicative variants
filterResult(contrib[[1]])
filterResult(contrib)
```

GenotypeMatrix-class *Class* GenotypeMatrix

Description

S4 class for storing genotypes efficiently as column-oriented sparse matrices along with variant info

Details

This class stores genotypes as a column-oriented sparse numeric matrix, where rows correspond to samples and columns correspond to variants. This is accomplished by extending the dgCMatrix class from which this class inherits all slots. Information about variants is stored in an additional slot named variantInfo. This slot must be of class VariantInfo and have exactly as many elements as the genotype matrix has columns. The variantInfo slot has a dedicated metadata column named "MAF" that contains the minor allele frequencies (MAFs) of the variants. For convenience, accessor functions variantInfo and MAF are available (see below).

Objects of this class should only be created and manipulated by the constructors and accessors described below, as only these methods ensure the integrity of the created objects. Direct modification of object slots is strongly discouraged!

Constructors

See help pages genotypeMatrix and readGenotypeMatrix.

Methods

show signature(object="GenotypeMatrix"): displays the matrix dimensions (i.e. the number of samples and variants) along with some basic statistics of the minor allele frequency (MAF).

Accessors

variantInfo signature(object="GenotypeMatrix"): returns variant information as a VariantInfo object.

MAF signature(object="GenotypeMatrix"): returns a numeric vector with the minor allele frequencies (MAFs).

Row and column names can be set and get as usual for matrix-like objects with rownames and colnames, respectively. When setting the column names of a GenotypeMatrix object, both the names of the variant info (slot variantInfo) and the column names of the matrix are set.

Subsetting

In the following code snippets, x is a GenotypeMatrix object.

x[i,] returns a GenotypeMatrix object that only contains the samples selected by the index vector i

x[, j] returns a GenotypeMatrix object that only contains the variants selected by the index vector j

x[i, j] returns a GenotypeMatrix object that only contains the samples selected by the index vector i and the variants selected by the index vector j

None of these subsetting functions support a drop argument. As soon as a drop argument is supplied, no matter whether TRUE or FALSE, all variant information is stripped off and a dgCMatrix object is returned.

By default, MAFs are not altered by subsetting samples. However, if the optional argument recomputeMAF is set to TRUE (the default is FALSE), MAFs are recomputed for the resulting subsetted genotype matrix as described in genotypeMatrix. The ploidy for computing MAFs can be controlled by the optional ploidy argument (the default is 2).

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

dgCMatrix, VariantInfo, genotypeMatrix, readGenotypeMatrix

Examples

```
## create a toy example
A <- matrix(rbinom(50, 2, prob=0.2), 5, 10)
sA <- as(A, "dgCMatrix")
pos <- sort(sample(1:10000, ncol(A)))
seqname <- "chr1"

## variant with 'GRanges' object
gr <- GRanges(seqnames=seqname, ranges=IRanges(start=pos, width=1))
gtm <- genotypeMatrix(A, gr)
gtm
as.matrix(gtm)
variantInfo(gtm)
MAF(gtm)

## variant with 'pos' and 'seqnames' object
genotypeMatrix(sA, pos, seqname)

## variant with 'seqname:pos' strings passed through 'pos' argument
```

```
spos <- paste(seqname, pos, sep=":")
spos
genotypeMatrix(sA, spos)

## read data from VCF file using 'readVcf()' from the 'VariantAnnotation'
## package
if (require(VariantAnnotation))
{
    vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
    sp <- ScanVcfParam(info=NA, genome="GT", fixed=c("ALT", "FILTER"))
    vcf <- readVcf(vcfFile, genome="hgA", param=sp)
    rowRanges(vcf)

    ## call constructor for 'VCF' object
    gtm <- genotypeMatrix(vcf)
    gtm
    variantInfo(gtm)

    ## alternatively, extract information from 'VCF' object and use
    ## variant with character matrix and 'GRanges' positions
    ## note that, in 'VCF' objects, rows correspond to variants and
    ## columns correspond to samples, therefore, we have to transpose the
    ## genotype
    gt <- t(geno(vcf)$GT)
    gt[1:5, 1:5]
    gr <- rowRanges(vcf)
    gtm <- genotypeMatrix(gt, gr)
    as.matrix(gtm[1:20, 1:5, recomputeMAF=TRUE])
}
```

---

genotypeMatrix-methods

Constructors for Creating GenotypeMatrix *Objects*

Description

Create GenotypeMatrix object from (sparse) matrix object and positions of variants

Usage

```
## S4 method for signature 'ANY,GRanges,missing'
genotypeMatrix(Z, pos, seqnames,
       ploidy=2, na.string=NULL, na.limit=1, MAF.limit=1,
       na.action=c("impute.major", "omit", "fail"),
       MAF.action=c("invert", "omit", "ignore", "fail"),
       sex=NULL)
## S4 method for signature 'ANY,numeric,character'
genotypeMatrix(Z, pos, seqnames, ...)
## S4 method for signature 'ANY,character,missing'
```

```
genotypeMatrix(Z, pos, seqnames, ...)
## S4 method for signature 'ANY,missing,missing'
genotypeMatrix(Z, pos, seqnames, subset,
       noIndels=TRUE, onlyPass=TRUE, sex=NULL, ...)
## S4 method for signature 'eSet,numeric,character'
genotypeMatrix(Z, pos, seqnames, ...)
## S4 method for signature 'eSet,character,missing'
genotypeMatrix(Z, pos, seqnames, ...)
## S4 method for signature 'eSet,character,character'
genotypeMatrix(Z, pos, seqnames, ...)
```

Arguments

Z	an object of class dgCMatrix, a numeric matrix, a character matrix, an object of class VCF, or an object of class eSet (see details below)
pos	an object of class GRanges, a numeric vector, or a character vector (see details below)
seqnames	a character vector (see details below)
ploidy	determines the ploidy of the genome for the computation of minor allele frequencies (MAFs) and the possible inversion of columns with an MAF exceeding 0.5; the elements of Z may not exceed this value.
subset	a numeric vector with indices or a character vector with names of samples to restrict to
na.limit	all columns with a missing value ratio above this threshold will be omitted from the output object.
MAF.limit	all columns with an MAF above this threshold will be omitted from the output object.
na.action	if "impute.major", all missing values will be imputed by major alleles in the output object. If "omit", all columns containing missing values will be omitted in the output object. If "fail", the function stops with an error if Z contains any missing values.
MAF.action	if "invert", all columns with an MAF exceeding 0.5 will be inverted in the sense that all minor alleles will be replaced by major alleles and vice versa. For numerical Z, this is accomplished by subtracting the column from the ploidy value. If "omit", all columns with an MAF greater than 0.5 are omitted in the output object. If "ignore", no action is taken and MAFs greater than 0.5 are kept as they are. If "fail", the function stops with an error if Z contains any column with an MAF greater than 0.5.
noIndels	if TRUE (default), only single nucleotide variants (SNVs) are considered and indels are skipped; only works if the ALT column is present in the VCF object Z, otherwise a warning is shown and the noIndels argument is ignored.
onlyPass	if TRUE (default), only variants are considered whose value in the FILTER column is "PASS"; only works if the FILTER column is present in the VCF object Z, otherwise a warning is shown and the onlyPass argument is ignored.
na.string	if not NULL, all "." entries in the character matrix or VCF genotype are replaced with this string before parsing the matrix.

sex	if NULL, all rows of Z are treated the same without any modifications; if sex is a factor with levels F (female) and M (male) that is as long as Z has rows, this argument is interpreted as the sex of the samples. In this case, the rows corresponding to male samples are doubled before further processing. This is designed for mixed-sex analyses of the X chromosome outside of the pseudoautosomal regions.
...	all additional arguments are passed on internally to the genotypeMatrix method with signature ANY,GRanges,missing.

Details

This method provides different ways of constructing an object of class GenotypeMatrix from other types of objects. The typical case is when a matrix object is combined with positional information. The first three variants listed above work with Z being a dgCMatrix object, a numeric matrix, or a character matrix.

If Z is a dgCMatrix object or a matrix, rows are interpreted as samples and columns are interpreted as variants. For dgCMatrix objects and numeric matrices, matrix entries are interpreted as the numbers of minor alleles (with 0 meaning only major alleles). In this case, minor allele frequencies (MAFs) are computed as column sums divided by the number of alleles, i.e. the number of samples/rows multiplied by the ploidy parameter. If Z is a character matrix, the matrix entries need to comply to the format of the "GT" field in VCF files. MAFs are computed as the actual relative frequency of minor alleles among all alleles in a column. For a diploid genome, therefore, this results in the same MAF estimate as mentioned above. However, some VCF readers, most importantly readVcf from the **VariantAnnotation** package, replace missing genotypes by a single "." even for non-haploid genomes, which would result in a wrong MAF estimate. To correct for this, the na.string parameter is available. If not NULL, all "." entries in the matrix are replaced by na.string before parsing the matrix. The correct setting for a diploid genome would be "./.".

Positional information can be passed to the function in three different ways:

- by supplying a GRanges object as pos argument and omitting the seqnames argument,
- by supplying a numeric vector of positions as pos argument and sequence/chromosome names as seqnames argument, or
- by supplying a character vector with entries of the format "seqname:pos" as pos argument and omitting the seqnames argument.

In all three cases, the lengths of the arguments pos and seqnames (if not omitted) must match the number of columns of Z.

If the arguments pos and seqnames are not specified, argument Z can (and must) be an object of class VCF (cf. package **VariantAnnotation**). In this case, the genotypeMatrix method extracts both the genotype matrix and positional information directly from the VCF object. Consequently, the VCF object Z must contain genotype information. If so, the genotype matrix is parsed and converted as described above for character matrices. Moreover, indels and variants that did not pass all quality filters can be skipped (see description of arguments noIndels and onlyPass above).

For all variants, filters in terms of missing values and MAFs can be applied. Moreover, variants with MAFs greater than 0.5 can filtered out or inverted. For details, see descriptions of parameters na.limit, MAF.limit, na.action, and MAF.action above.

For convenience, genotypeMatrix also allows for converting SNP genotype matrices stored in eSet objects, e.g. SnpSet objects or SnpSetIllumina objects (cf. package **beadarraySNP**). If genotypeMatrix is called with an eSet object as first argument Z, the method first checks whether there is a slot call in assayData(Z) and whether it is a matrix. If so, this matrix is interpreted as follows: 1 corresponds to genotype "AA", 2 corresponds to the genotype "Aa", and 3 corresponds to the genotype "aa", where "A" is the major allele and "a" is the minor allele. If pos is a numeric vector and seqnames is a character vector or if pos is a character vector and seqnames is missing, then these two arguments are interpreted as described above. However, if pos and seqnames are both single strings (character vectors of length 1), then pos is interpreted as the name of the feature data column that contains positional information and seqnames is interpreted as the feature data column that contains the chromosome on which each variant is located. Correspondingly, featureData(Z)[[pos]] must be available and must be a numeric vector. Correspondingly, featureData(Z)[[seqnames]] must be available and must be a character vector (or a data type that can be cast to a character vector).

Value

returns an object of class GenotypeMatrix

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

https://github.com/samtools/hts-specs

Obenchain, V., Lawrence, M., Carey, V., Gogarten, S., Shannon, P., and Morgan, M. (2014) VariantAnnotation: a Bioconductor package for exploration and annotation of genetic variants. *Bioinformatics* **30**, 2076-2078. DOI: doi:10.1093/bioinformatics/btu168.

See Also

GenotypeMatrix, dgCMatrix, GRanges

Examples

```
## create a toy example
A <- matrix(rbinom(50, 2, prob=0.2), 5, 10)
sA <- as(A, "dgCMatrix")
pos <- sort(sample(1:10000, ncol(A)))
seqname <- "chr1"

## variant with 'GRanges' object
gr <- GRanges(seqnames=seqname, ranges=IRanges(start=pos, width=1))
gtm <- genotypeMatrix(A, gr)
gtm
as.matrix(gtm)
variantInfo(gtm)
MAF(gtm)
```

```
## variant with 'pos' and 'seqnames' object
genotypeMatrix(sA, pos, seqname)

## variant with 'seqname:pos' strings passed through 'pos' argument
spos <- paste(seqname, pos, sep=":")
spos
genotypeMatrix(sA, spos)

## read data from VCF file using 'readVcf()' from the 'VariantAnnotation'
## package
if (require(VariantAnnotation))
{
    vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
    sp <- ScanVcfParam(info=NA, genome="GT", fixed=c("ALT", "FILTER"))
    vcf <- readVcf(vcfFile, genome="hgA", param=sp)
    rowRanges(vcf)

    ## call constructor for 'VCF' object
    gtm <- genotypeMatrix(vcf)
    gtm
    variantInfo(gtm)

    ## alternatively, extract information from 'VCF' object and use
    ## variant with character matrix and 'GRanges' positions
    ## note that, in 'VCF' objects, rows correspond to variants and
    ## columns correspond to samples, therefore, we have to transpose the
    ## genotype
    gt <- t(geno(vcf)$GT)
    gt[1:5, 1:5]
    gr <- rowRanges(vcf)
    gtm <- genotypeMatrix(gt, gr)
    as.matrix(gtm[1:20, 1:5, recomputeMAF=TRUE])
}
```

---

hgA *Artificial Human Chromosome for Testing Purposes*

Description

A GRanges object defining a minimalistic artificial human chromosome with 200,000 bp length

Usage

```
hgA
```

Format

Real human genome-based examples would require the supply of massive data and would require lengthy computation times. Therefore, the examples supplied with this package are based on a

small single-chromosome artificial genome. The GRanges object hgA provides a description of this artificial genome that can be used for further processing, e.g. by the partitionRegions function.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

GRanges, partitionRegions

Examples

```
## load data set
data(hgA)

## display hgA
show(hgA)
genome(hgA)

## partition hgA into overlapping regions of length 10,000 bp
partitionRegions(hgA, width=10000)
```

---

nullModel *Create Null Model for Association Test*

Description

Method for creating a null model that can be used for association testing using assocTest

Usage

```
## S4 method for signature 'formula,data.frame'
nullModel(X, y, data,
          type=c("automatic", "logistic", "linear", "bernoulli"),
          n.resampling=0,
          type.resampling=c("bootstrap", "permutation"),
          adj=c("automatic", "none", "force"), adjExact=FALSE,
          n.resampling.adj=10000, checkData=TRUE)
## S4 method for signature 'formula,missing'
nullModel(X, y, data,
          type=c("automatic", "logistic", "linear", "bernoulli"),
          n.resampling=0,
          type.resampling=c("bootstrap", "permutation"),
          adj=c("automatic", "none", "force"), adjExact=FALSE,
```

```
          n.resampling.adj=10000, checkData=TRUE)
## S4 method for signature 'matrix,numeric'
nullModel(X, y,
          type=c("automatic", "logistic", "linear"), ...)
## S4 method for signature 'matrix,factor'
nullModel(X, y,
          type=c("automatic", "logistic", "linear"), ...)
## S4 method for signature 'missing,numeric'
nullModel(X, y,
          type=c("automatic", "logistic", "linear", "bernoulli"),
          ...)
## S4 method for signature 'missing,factor'
nullModel(X, y,
          type=c("automatic", "logistic", "linear", "bernoulli"),
          ...)
```

Arguments

X	a formula or matrix
y	if the formula interface is used, y can be used to pass a data frame with the table in which both covariates and traits are contained (alternatively, the data argument can be used for that purpose). The other methods (if X is not a formula) expect y to be the trait vector. Trait vectors can either be numeric vectors or a factor with two levels (see details below).
data	for consistency with standard R methods from the **stats** package, the data frame can also be passed to nullModel via the data argument. In this case, the y must be empty. If y is specified, data is ignored.
type	type of model to train (see details below)
n.resampling	number of null model residuals to sample; set to zero (default) to turn resampling off; resampling is not supported for plain trait vectors without covariates
type.resampling	method how to sample null model residuals; the choice "permutation" refers to simple random permutations of the model's residuals. If "bootstrap" is chosen (default), the following strategy is applied for linear models (continuous trait): residuals are sampled as normally distributed values with mean 0 and the same standard deviation as the model's residuals. For logistic models (binary trait), the choice "bootstrap" selects the same bootstrapping method that is implemented in the **SKAT** package.
adj	whether or not to use small sample correction for logistic models (binary trait with covariates). The choice "none" turns off small sample correction. If "force" is chosen, small sample correction is turned on unconditionally. If "automatic" is chosen (default), small sample correction is turned on if the number of samples does not exceed 2,000. This argument is ignored for any type of model except "logistic" and small sample correction is switched off.
adjExact	in case small sample correction is switched on (see above), this argument indicates whether or not the exact square root of the matrix P_0 should be precomputed (see Subsection 9.5 of the package vignette). The default is FALSE. This argument is ignored if small sample correction is not switched on.

`n.resampling.adj`
: number of null model residuals to sample for the adjustment of higher moments; ignored if small sample correction is switched off.

`checkData`
: if `FALSE`, only a very limited set of input checks is performed. The purpose of this option is to save computational effort for repeated input checks if the function is called from a function that has already performed input checks. The default is `TRUE`. Only change to `FALSE` if you know what you are doing!

`...`
: all other parameters are passed on to the `nullModel` method with signature `formula,data.frame`.

Details

The **podkat** package assumes a mixed model in which the trait under investigation depends both on covariates (if any) and the genotype. The `nullModel` method models the relationship between the trait and the covariates (if any) without taking the genotype into account, which corresponds to the null assumption that the trait and the genotype are independent. Therefore, we speak of *null models*. The following types of models are presently available:

Linear model (type "linear"): a linear model is trained for a continuous trait and a given set of covariates (if any); this is done by standard linear regression using the `lm` function.

Logistic linear model (type "logistic"): a generalized linear model is trained for a binary trait and a given set of covariates (if any); this is done by logistic regression using the `glm` function.

Bernoulli-distributed trait (type "bernoulli"): a binary trait without covariates is interpreted as instances of a simple Bernoulli process with p being the relative frequencies 1's/cases.

The `type` argument can be used to select the type of model, where the following restrictions apply:

- For linear models, the trait vector must be numerical. Factors/factor columns are not accepted.
- For logistic models and Bernoulli-distributed traits, both numerical vectors and factors are acceptable. In any case, only 0's (controls) and 1's (cases) are accepted. Furthermore, `nullModel` quits with an error if the trait shows no variation. In other words, trait vectors that only contain 0's or only contain 1's are not accepted (as association testings makes little sense for such traits anyway).

The following interfaces are available to specify the traits and the covariates (if any):

Formula interface: the first argument `X` can be a formula that specifies the trait vector/column, the covariate matrix/columns (if any), and the intercept (if any). If neither the `y` argument nor the `data` argument is specified, `nullModel` searches the environment from which the function has been called. This interface is largely analogous to the functions `lm` and `glm`.

Trait vector without covariates: if the `X` argument is omitted and `y` is a numeric vector or factor, `y` is interpreted as trait vector, and a null model is created from `y` without covariates. Linear and logistic models are trained with an intercept. For type "bernoulli", the trait vector is written to the output object as is.

Trait vector plus covariate matrix: if the `X` argument is a matrix and `y` is a numeric vector or factor, `y` is interpreted as trait vector and `X` is interpreted as covariate matrix. In this case, linear and logistic models are trained as (generalized) linear regressors that predict the trait from the covariates plus an intercept. The type "bernoulli" is not available for this variant, since this type of model cannot consider covariates.

All `nullModel` methods also support the choice `type="automatic"`. In this case, `nullModel` guesses the most reasonable type of model in the following way: If the trait vector/column is a factor or a numeric vector containing only 0's and 1's (where both values must be present, as noted above already), the trait is supposed to be binary and the type "logistic" is assumed, unless the following conditions are satisfied:

1. The number of samples does not exceed 100.
2. No intercept and no covariates have been specified. This condition can be met by supplying an empty model to the formula interface (e.g. `y ~ 0`) or by supplying the trait vector as argument `y` while omitting `X`.

If these two conditions are fulfilled for a binary trait, `nullModel` chooses the type "bernoulli". If the trait is not binary and the trait vector/column is numeric, `nullModel` assumes type "linear".

For consistency with the **SKAT** package, the **podkat** package also offers *resampling*, i.e. a certain number of vectors of residuals are sampled according to the null model. This can be done when training the null model by setting the `n.resampling` parameter (number of residual vectors that are sampled) to a value larger than 0. Then, when association testing is performed, p-values are computed also for all these sampled residuals, and an additional estimated p-value is computed as the relative frequency of p-values of sampled residuals that are at least as significant as the test's p-value. The procedure to sample residuals is controlled with the `type.resampling` argument (see above).

For logistic models (type "logistic"), `assocTest` offers the small sample correction as introduced by *Lee et al. (2012)*. If the adjustment of higher moments should be applied, some preparations need to be made already when training the null model. Which preparations are carried out, can be controlled by the arguments `adj`, `adjExact`, `n.resampling.adj`, and `type.resampling` (see descriptions of arguments above and Subsection 9.5 of the package vignette).

If any missing values are found in the trait vector/column or the covariate matrix/columns, the respective samples are omitted from the resulting model (which is the standard behavior of `lm` and `glm` anyway). The indices of the omitted samples are stored in the `na.omit` slot of the returned `NullModel` object.

Value

returns a `NullModel` object

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

Lee, S., Emond, M. J., Bamshad, M. J., Barnes, K. C., Rieder, M. J., Nickerson, D. A., NHLBI Exome Sequencing Project - ESP Lung Project Team, Christiani, D. C., Wurfel, M. M., and Lin, X. (2012) Optimal unified approach for rare-variant association testing with application to small-sample case-control whole-exome sequencing studies. *Am. J. Hum. Genet.* **91**, 224-237. DOI: doi:10.1016/j.ajhg.2012.06.007.

See Also

NullModel, lm, glm

Examples

```
## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
model <- nullModel(y ~ ., pheno)
model
length(model)
residuals(model)

## read phenotype data from CSV file (binary trait + covariates)
phenoFile <- system.file("examples/example1log.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
model <- nullModel(y ~ ., pheno)
model
length(model)
residuals(model)

## "train" simple Bernoulli model on a subset of 100 samples
model <- nullModel(y ~ 0, pheno[1:100, ])
model
length(model)
residuals(model)

## alternatively, use the interface that only supplies the
## trait vector
model <- nullModel(y=pheno[1:100, ]$y)
model
```

NullModel-class *Class* NullModel

Description

S4 class for storing null models for later usage with the assocTest method

Objects

Objects of this class are created by calling nullModel.

Slots

The following slots are defined for `NullModel` objects:

`type`: type of model

`residuals`: residuals of linear model; for type "bernoulli", this is simply the trait vector (see `nullModel-methods` for details)

`model.matrix`: model matrix of the (generalized) linear model trained for the covariates (if any)

`inv.matrix`: pre-computed inverse of some matrix needed for computing the null distribution; only used for types "logistic" and "linear"

`P0sqrt`: pre-computed square root of matrix P_0 (see Subsections 9.1 and 9.5 of the package vignette); needed for computing the null distribution in case the small sample correction is used for a logistic model; computed only if `nullModel` is called with `adjExact=TRUE`.

`coefficients`: coefficients of (generalized) linear model trained for the covariates (if any)

`na.omit`: indices of samples omitted from (generalized) linear model because of missing values in target or covariates

`n.cases`: for binary traits (types "logistic" and "bernoulli"), the number of cases, i.e. the number of 1's in the trait vector

`variance`: for continuous traits (type "linear"), this is a single numeric value with the variance of residuals of the linear model; for logistic models with binary traits (type "logistic"), this is a vector with variances of the per-sample Bernoulli distributions; for later use of the exact mixture-of-Bernoulli test (type "bernoulli"), this is the variance of the Bernoulli distribution

`prob`: for logistic models with binary traits (type "logistic"), this is a vector with probabilities of the per-sample Bernoulli distributions; for later use of the exact mixture-of-Bernoulli test (type "bernoulli"), this is the probability of the Bernoulli distribution

`type.resampling`: which resampling algorithm was used

`res.resampling`: matrix with residuals sampled under the null hypothesis (if any)

`res.resampling.adj`: matrix with residuals sampled under the null hypothesis for the purpose of higher moment correction (if any; only used for logistic models with small sample correction)

`call`: the matched call with which the object was created

Details

This class serves as the general interface for storing the necessary phenotype information for a later association test. Objects of this class should only be created by the `nullModel` function. Direct modification of object slots is strongly discouraged!

Methods

show `signature(object="NullModel")`: displays basic information about the null model, such as, the type of the model and the numbers of covariates.

Accessors

residuals `signature(object="NullModel")`: returns the `residuals` slot.

names `signature(object="NullModel")`: returns the names of samples in the null model.

coefficients `signature(object="NullModel")`: returns the `coefficients` slot.

length `signature(x="NullModel")`: returns the number of samples that was used to train the null model.

Subsetting

For a `NullModel` object `x` and an index vector `i` that is a permutation of `1:length(x)`, `x[i]` returns a new `NullModel` object in which the samples have been rearranged according to the permutation `i`. This is meant for applications in which the order of the samples in a subsequent association test is different from the order of the samples when the null model was trained/created.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

nullModel

Examples

```
## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
model <- nullModel(y ~ ., pheno)
model
length(model)
residuals(model)

## read phenotype data from CSV file (binary trait + covariates)
phenoFile <- system.file("examples/example1log.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
model <- nullModel(y ~ ., pheno)
model
length(model)
residuals(model)

## "train" simple Bernoulli model on a subset of 100 samples
model <- nullModel(y ~ 0, pheno[1:100, ])
```

```
model
length(model)
residuals(model)
```

`p.adjust-methods` *Adjust p-Value for Multiple Tests*

Description

Given an `AssocTestResultRanges` object, this method adds a metadata column with adjusted p-values.

Usage

```
## S4 method for signature 'AssocTestResultRanges'
p.adjust(p, method=p.adjust.methods, n=length(p))
```

Arguments

`p`	object of class `AssocTestResultRanges`
`method`	correction method (see `p.adjust.methods`)
`n`	parameter available for consistency with standard `p.adjust` function; ignored in this implementation

Details

This function is a wrapper around the standard `p.adjust` function from the **stats** package. It takes the `p.value` metadata column from the `AssocTestResultRanges` object `p`, applies the multiple testing correction method specified as `method` argument. The method returns a copy of `p` with an additional metadata column `p.value.adj` that contains the adjusted p-values. If `p` already contained a metadata column `p.value.adj`, this column is overwritten with the new adjusted p-values.

If `p` also contains a metadata column `p.value.resampled`, multiple testing correction is also applied to resampled p-values. The resulting adjusted p-values are placed in the metadata column `p.value.resampled.adj`.

Note that, for consistency with the standard `p.adjust` function, the default correction method is "holm".

Value

an `AssocTestResultRanges` object (see details above)

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

AssocTestResultRanges, p.adjust

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.lin <- nullModel(y ~ ., pheno)

## perform association test for multiple regions
res <- assocTest(Z, nm.lin, windows)
print(res)

## perform multiple testing correction
res.adj <- p.adjust(res, method="BH")
print(res.adj)
```

---

partitionRegions-methods

Partition Genomic Regions

Description

Partitions genomic regions into windows of fixed size

Usage

```
## S4 method for signature 'GRanges'
partitionRegions(x, chrs=character(), width=5000, overlap=0.5)
## S4 method for signature 'GRangesList'
partitionRegions(x, chrs=character(), width=5000, overlap=0.5)
## S4 method for signature 'MaskedBSgenome'
partitionRegions(x, chrs=character(), width=5000, overlap=0.5, ...)
```

Arguments

x	an object of class GRanges, GRangesList, or MaskedBSgenome
chrs	a character vector (possibly empty) with names of chromosomes to limit to
width	window size
overlap	amount of overlap; a zero value corresponds to non-overlapping windows and the default 0.5 corresponds to 50% overlap. The largest possible value is 0.8 which corresponds to an overlap of 80%.
...	further arguments are passed on to unmaskedRegions.

Details

For a GRanges object x, this method partitions each genomic region into possibly overlapping, equally large windows of size width. The amount of overlap is controlled with the overlap parameter. The windows are placed such that possible overhangs are balanced at the beginning and end of the region. As an example, suppose we have a region from bases 1 to 14,000 and that we want to cover it with windows of 10,000bp length and 50% overlap. The straightforward approach would be to have two windows 1-10,000 and 5,001-15,000, and to crop the latter to 5,001-14,000. As said, the partitionRegions balances the overhangs, so it will return two windows 1-9,500 and 4,501-14,000 instead.

If chrs is not empty, partitionRegions will only consider regions from those chromosomes (i.e. regions in the GRanges object whose seqnames occur in chrs).

If called for a GRangesList object, all componentes of the GRangesList object are partitioned separately as described above.

For convenience, this function can also be called for a MaskedBSgenome object. In this case, unmaskedRegions is called before partitioning.

Value

If x is a GRanges object, the function also returns a GRanges object. In the other two cases, a GRangesList object is returned.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

assocTest, unmaskedRegions, unmasked-datasets, GRangesList, GRanges

Examples

```
## create a toy example
gr <- GRanges(seqnames="chr1", ranges=IRanges(start=1, end=14000))
partitionRegions(gr, width=10000, overlap=0.5)
partitionRegions(gr, width=10000, overlap=0.8)
partitionRegions(gr, width=10000, overlap=0)

## a toy example of a 'GRangesList'
grL <- GRangesList(gr, GRanges(seqnames="chr2",
                               ranges=IRanges(start=1, end=22000)))
partitionRegions(grL, width=10000, overlap=0.5)
partitionRegions(grL, width=10000, overlap=0.8)

## real-world example
data(hg38Unmasked)
partitionRegions(hg38Unmasked, chrs="chr22", width=20000)
```

---

plot *Plotting functions*

Description

Functions for visualizing association test results by means of a Manhattan plot and for visualizing genotypes

Usage

```
## S4 method for signature 'AssocTestResultRanges,missing'
plot(x, y, cutoff=0.05,
     which=c("p.value", "p.value.adj", "p.value.resampled",
             "p.value.resampled.adj"), showEmpty=FALSE,
     as.dots=FALSE, pch=19, col=c("darkgrey", "grey"), scol="red",
     lcol="red", xlab=NULL, ylab=NULL, ylim=NULL, lwd=1, cex=1,
     cexXaxs=1, cexYaxs=1, srt=0, adj=c(0.5, 1), ...)
## S4 method for signature 'AssocTestResultRanges,character'
plot(x, y, cutoff=0.05,
     which=c("p.value", "p.value.adj", "p.value.resampled",
             "p.value.resampled.adj"), showEmpty=FALSE,
     as.dots=FALSE, pch=19, col=c("darkgrey", "grey"), scol="red",
     lcol="red", xlab=NULL, ylab=NULL, ylim=NULL, lwd=1, cex=1,
     cexXaxs=1, cexYaxs=1, srt=0, adj=c(0.5, 1), ...)
## S4 method for signature 'AssocTestResultRanges,GRanges'
plot(x, y, cutoff=0.05,
     which=c("p.value", "p.value.adj", "p.value.resampled",
             "p.value.resampled.adj"), showEmpty=FALSE,
     as.dots=FALSE, pch=19, col="darkgrey", scol="red", lcol="red",
     xlab=NULL, ylab=NULL, ylim=NULL, lwd=1, cex=1,
     cexXaxs=1, cexYaxs=1, ...)
```

```
## S4 method for signature 'GenotypeMatrix,missing'
plot(x, y, col="black",
     labRow=NULL, labCol=NULL, cexXaxs=(0.2 + 1 / log10(ncol(x))),
     cexYaxs=(0.2 + 1 / log10(nrow(x))), srt=90, adj=c(1, 0.5))
## S4 method for signature 'GenotypeMatrix,factor'
plot(x, y, col=rainbow(length(levels(y))),
     labRow=NULL, labCol=NULL, cexXaxs=(0.2 + 1 / log10(ncol(x))),
     cexYaxs=(0.2 + 1 / log10(nrow(x))), srt=90, adj=c(1, 0.5))
## S4 method for signature 'GenotypeMatrix,numeric'
plot(x, y, col="black", ccol="red", lwd=2,
     labRow=NULL, labCol=NULL, cexXaxs=(0.2 + 1 / log10(ncol(x))),
     cexYaxs=(0.2 + 1 / log10(nrow(x))), srt=90, adj=c(1, 0.5))
## S4 method for signature 'GRanges,character'
plot(x, y, alongGenome=FALSE,
     type=c("r", "s", "S", "l", "p", "b", "c", "h", "n"),
     xlab=NULL, ylab=NULL, col="red", lwd=2,
     cexXaxs=(0.2 + 1 / log10(length(x))), cexYaxs=1,
     frame.plot=TRUE, srt=90, adj=c(1, 0.5), ...)
```

Arguments

x	an object of class AssocTestResultRanges, GenotypeMatrix, or GRanges
y	a character string, GRanges object, or factor
cutoff	significance threshold
which	a character string specifying which p-values to plot; if "p.value" (default), raw p-values are plotted. Other options are "p.value.adj" (adjusted p-values), "p.value.resampled" (resampled p-values), and "p.value.resampled.adj" (adjusted resampled p-values). If the requested column is not present in the input object x, the function stops with an error message.
showEmpty	if FALSE (default), p-values of regions that did not contain any variants are omitted from the plot.
as.dots	if TRUE, p-values are plotted as dots/characters in the center of the genomic region. If FALSE (default), p-values are plotted as lines stretching from the starts to the ends of the corresponding genomic regions.
pch	plotting character used to plot a single p-value, ignored if as.dots=FALSE; see points for details.
col	plotting color(s); see details below
scol	color for plotting significant p-values (i.e. the ones passing the significance threshold)
lcol	color for plotting the significance threshold line
xlab	x axis label; if NULL (default) or NA, plot makes an automatic choice
ylab	y axis label; if NULL (default) or NA, plot makes an automatic choice
ylim	y axis limits; if NULL (default) or NA, plot makes an automatic choice; if user-specified, ylim must be a two-element numeric vector with the first element being 0 and the second element being a positive value.

lwd	line thickness; in Manhattan plots, this parameter corresponds to the thickness of the significance threshold line. When plotting genotype matrices along with continuous traits, this is the thickness of the line that corresponds to the trait.
cex	scaling factor for plotting p-values; see points for details.
labRow, labCol	row and column labels; set to NA to switch labels off; if NULL, rows are labeled by sample names (rownames(x)) and columns are labeled by variant names (names(variantInfo(x))).
cexXaxs, cexYaxs	scaling factors for axes labels
ccol	color of the line that plots the continuous trait along with a genotype matrix
srt	rotation angle of text labels on horizontal axis (see text for details); ignored if standard numerical ticks and labels are used.
adj	adjustment of text labels on horizontal axis (see text for details); ignored if standard numerical ticks and labels are used.
alongGenome	plot along the genome or per region (default); see details below.
type	type of plot; see plot.default for details. Additionally, the type "r" is available (default) which plots horizontal lines along the regions of x.
frame.plot	whether or not to frame the plotting area (see plot; default: TRUE)
...	all other arguments are passed to plot.

Details

If plot is called for an AssocTestResultRanges object without specifying the second argument y, a so-called Manhattan plot is produced. The x axis corresponds to the genome on which the AssocTestResultRanges x is based and the y axis shows absolute values of log-transformed p-values. The which argument determines which p-value is plotted, i.e. raw p-values, adjusted p-values, resampled p-values, or adjusted resampled p-values. The cutoff argument allows for setting a significance threshold above which p-values are plotted in a different color (see above).

The optional y argument can be used for two purposes: (1) if it is a character vector containing chromosome names (sequence names), it can be used for specifying a subset of one or more chromosomes to be plotted. (2) if y is a GRanges object of length 1 (if longer, plot stops with an error), only the genomic region corresponding to y is plotted.

The col argument serves for specifying the color for plotting insignificant p-values (i.e. the ones above the significance threshold); if the number of colors is smaller than the number of chromosomes, the vector is recycled. If col is a single color, all insignificant p-values are plotted in the same color. If col has two elements (like the default value), the insignificant p-values of different chromosomes are plotted with alternating colors. It is also possible to produce density plots of p-values by using semi-transparent colors (see, e.g., rgb or hsv for information on how to use the alpha channel).

If plot is called for a GenotypeMatrix object x and no y argument, the matrix is visualized in a heatmap-like fashion, where two major alleles are displayed in white, two minor alleles are displayed in the color passed as col argument, and the heterozygotous case (one minor, one major) is displayed in the color passed as col argument, but with 50% transparency. The arguments cexYaxs and cexXaxs can be used to change the scaling of the axis labels.

If plot is called for a GenotypeMatrix object x and a factor y, then the factor y is interpreted as a binary trait. In this case, the rows of the genotype matrix x are reordered such that rows/samples with the same label are plotted next to each other. Each such group can be plotted in a different color. For this purpose, a vector of colors can be passed as col argument.

If plot is called for a GenotypeMatrix object x and a numeric vector y, then the vector y is interpreted as a continuous trait. In this case, the rows of the genotype matrix x are reordered according to the trait vector y and the genotype matrix is plotted as described above. The trait y is superimposed in the plot in color ccol and with line width lwd. If the null model has been trained with covariates, it also makes sense to plot the genotype against the null model residuals, since these are exactly the values that the genotypes were tested against.

If plot is called for a GRanges object x and a character string y, then plot checks whether x has a metadata column named y. If it exists, this column is plotted against the regions in x. If alongGenome is FALSE (which is the default), the regions in x are arranged along the horizontal plot axis with equal widths and in the same order as contained in x. If the regions in x are named, then the names are used as axis labels and the argument cexXaxs can be used to scale the font size of the names. If alongGenome is TRUE, the metadata column is plotted against genomic positions. The knots of the curves are then positioned at the positions given in the GRanges object x. For types "s", "S", "l", "p", "b", "c", and "h", knots are placed in the middle of the genomic regions contained in x if they are longer than one base. For type "r", regions are plotted as lines exactly stretching between the start and end coordinates of each region in x.

Value

returns an invisible numeric vector of length 2 containing the y axis limits

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

AssocTestResultRanges, GRanges

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## plot some fraction of the genotype matrix
```

```
plot(Z[, 1:25])

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1log.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.log <- nullModel(y ~ ., pheno)

## perform association test
res <- assocTest(Z, nm.log, windows)
res.adj <- p.adjust(res)

## plot results
plot(res)
plot(res, cutoff=1.e-5, as.dots=TRUE)
plot(res.adj, which="p.value.adj")
plot(res.adj, reduce(windows[3:5]), which="p.value.adj")

## filter regions
res.adj.f <- filterResult(res.adj, filterBy="p.value.adj")

## plot genotype grouped by target
sel <- which(overlapsAny(variantInfo(Z), reduce(res.adj.f)))
plot(Z[, sel], factor(pheno$y))
plot(Z[, sel], residuals(nm.log), srt=45)

## compute contributions
contrib <- weights(res.adj.f, Z, nm.log)
contrib[[1]]

## plot contributions
plot(contrib[[1]], "weight.raw")
plot(contrib[[1]], "weight.contribution", type="b", alongGenome=TRUE)
```

`print-methods` *Print Association Test Results*

Description

Display method for S4 class `AssocTestResultRanges`

Usage

```
## S4 method for signature 'AssocTestResultRanges'
print(x, cutoff=0.05,
        sortBy=c("p.value", "p.value.adj", "p.value.resampled",
                 "p.value.resampled.adj", "genome", "none"),
        max.show=10)
```

Arguments

x	an object of class AssocTestResultRanges
cutoff	a numerical vector with one or more p-value thresholds; if present (otherwise NA or an empty vector must be passed), print displays the number of tested regions with a p-value below each threshold. If the AssocTestResultRanges object also contains adjusted p-values, the numbers of tested regions with p-values below each of the thresholds are printed too. If max.show is greater than 0, the max.show most significant regions up to an (adjusted) p-value (depending on the sortBy argument) up to the largest threshold are shown.
sortBy	a character string that determines (1) how regions are sorted and (2) according to which p-value the cutoff threshold is applied when printing regions; if sortBy is "p.value" (default), regions are sorted according to raw p-values and only the max.show most significant regions are printed - as long as the raw p-value is not larger than the largest value in the cutoff argument. For "p.value.adj", regions are sorted and filtered according to adjusted p-values, analogously for choices "p.value.resampled" and "p.value.resampled.adj". In case that sortBy is "genome", the p-values are ignored and the first max.show regions in the genome are displayed. In case that sortBy is "none", the p-values are also ignored and the first max.show regions are displayed in the order as they appear in the AssocTestResultRanges object.
max.show	maximum number of regions to display; if 0, no regions are displayed at all.

Details

print displays the most important information stored in an AssocTestResultRanges object x. That includes the type of null model, the numbers of samples and tested regions, the kernel that was used for testing, etc. Depending on the cutoff argument, a certain number of significant tests is printed. If max.show is larger than 0, then some regions are shown along with association test results. Which regions are selected and how they are sorted, depends on the arguments sortBy and cutoff (see above).

Value

print returns its argument x invisibly.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

GenotypeMatrix, NullModel, AssocTestResult, AssocTestResultRanges

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.lin <- nullModel(y ~ ., pheno)

## perform association test for multiple regions
res <- assocTest(Z, nm.lin, windows)

## perform multiple testing correction
res.adj <- p.adjust(res)

## print 'AssocTestResultRanges' object
print(res)
print(res, max.show=0)
print(res.adj, cutoff=c(0.005, 0.01, 0.05))
print(res.adj, cutoff=0.05, sortBy="p.value.adj")
print(res.adj, cutoff=NA, sortBy="none", max.show=40)
```

---

qqplot *Quantile-Quantile Plots*

Description

Functions for visualizing association test results by means of a quantile-quantile (Q-Q) plot

Usage

```
## S4 method for signature 'AssocTestResultRanges,missing'
qqplot(x, y,
       xlab=deparse(substitute(x)), ylab=deparse(substitute(y)),
       common.scale=TRUE, preserveLabels=FALSE, lwd=1,
       lcol="red", ..., conf.level=NULL, conf.args=NULL)
## S4 method for signature 'AssocTestResultRanges,AssocTestResultRanges'
qqplot(x, y,
       xlab=deparse(substitute(x)), ylab=deparse(substitute(y)),
       common.scale=TRUE, preserveLabels=FALSE, lwd=1,
       lcol="red", ..., conf.level=NULL, conf.args=NULL)
```

Arguments

x, y	objects of class AssocTestResultRanges
xlab	if preserveLabels is TRUE, xlab is interpreted as axis label for the horizontal axis; if preserveLabels is FALSE, xlab can be a character string or expression that is interpreted as a name/label for the object x and is used for determining an appropriate axis label.
ylab	if preserveLabels is TRUE, ylab is interpreted as axis label for the vertical axis; if preserveLabels is FALSE, ylab can be a character string or expression that is interpreted as a name/label for the object y and is used for determining an appropriate axis label.
common.scale	if TRUE (default), the same plotting ranges are used for both axes; if FALSE, the two axes are scaled independently.
preserveLabels	if TRUE, xlab and ylab are used as axis labels without any change; if FALSE (default), the function interprets xlab and ylab as object labels for x and y and uses them for determining axis labels appropriately
lwd	line width for drawing the diagonal line which theoretically corresponds to the equality of the two distributions; if zero, no diagonal line is drawn.
lcol	color for drawing the diagonal line
conf.level	dummy argument for compatibility with the new version of the function in the **stats** package; currently unused
conf.args	dummy argument for compatibility with the new version of the function in the **stats** package; currently unused
...	all other arguments are passed to plot;

Details

If qqplot is called for an AssocTestResultRanges object without specifying the second argument y, a Q-Q plot of the raw p-values in x against a uniform distribution of expected p-values is created, where the theoretical p-values are computed using the ppoints function. In this case, the log-transformed observed p-values contained in x are plotted on the vertical axis and the log-transformed expected p-values are plotted on the horizontal axis. If preserveLabels is TRUE, xlab and ylab are used as axis labels as usual. However, if preserveLabels is FALSE, which is the default, xlab is interpreted as object label for x, i.e. the object whose p-values are plotted on the vertical axis.

If qqplot is called for two AssocTestResultRanges object x and y, the log-transformed raw p-values of x and y are plotted against each other, where the p-values of x are plotted on the horizontal axis and the p-values of x are plotted on the vertical axis.

Value

like the standard qqplot function from the **stats** package, qqplot returns an invisible list containing the two sorted vectors of p-values.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

AssocTestResultRanges

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.lin <- nullModel(y ~ ., pheno)

## perform association tests
res.p <- assocTest(Z, nm.lin, windows, kernel="linear.podkat")
res.s <- assocTest(Z, nm.lin, windows, kernel="linear.SKAT")

## plot results
qqplot(res.p)
qqplot(res.p, res.s, xlab="PODKAT results", ylab="SKAT results")
qqplot(res.p, res.s, xlab="PODKAT results", ylab="SKAT results",
       preserveLabels=TRUE)
qqplot(res.p, res.s, common.scale=FALSE)
```

---

readGenotypeMatrix-methods

Read from VCF File

Description

A fast lightweight function that reads from a VCF file and returns the result as a GenotypeMatrix object

Usage

```
## S4 method for signature 'TabixFile,GRanges'
readGenotypeMatrix(file, regions, subset,
                   noIndels=TRUE, onlyPass=TRUE,
                   na.limit=1, MAF.limit=1,
                   na.action=c("impute.major", "omit", "fail"),
                   MAF.action=c("invert", "omit", "ignore", "fail"),
                   sex=NULL)
## S4 method for signature 'TabixFile,missing'
readGenotypeMatrix(file, regions, ...)
## S4 method for signature 'character,GRanges'
readGenotypeMatrix(file, regions, ...)
## S4 method for signature 'character,missing'
readGenotypeMatrix(file, regions, ...)
```

Arguments

file	a TabixFile object or a character string with a file name of the VCF file to read from; if file is a file name, the method internally creates a TabixFile object for this file name.
regions	a GRanges object that specifies which genomic regions to read from the VCF file; if missing, the entire VCF file is read.
subset	a numeric vector with indices or a character vector with names of samples to restrict to; if specified, only these samples' genotypes are read from the VCF file and all other samples are ignored and omitted from the GenotypeMatrix object that is returned. Moreover, minor allele frequencies (MAFs) are only computed from the genotypes of the samples specified by subset.
noIndels	if TRUE (default), only single-nucleotide variants (SNVs) are considered and indels are skipped.
onlyPass	if TRUE (default), only variants are considered whose value in the FILTER column is "PASS".
na.limit	all variants with a missing value ratio above this threshold will be omitted from the output object.
MAF.limit	all variants with an MAF above this threshold will be omitted from the output object.
na.action	if "impute.major", all missing values will be imputed by major alleles in the output object. If "omit", all variants containing missing values will be omitted in the output object. If "fail", the function stops with an error if a variant contains any missing values.
MAF.action	if "invert", all variants with an MAF exceeding 0.5 will be inverted in the sense that all minor alleles will be replaced by major alleles and vice versa. If "omit", all variants with an MAF greater than 0.5 are omitted in the output object. If "ignore", no action is taken and MAFs greater than 0.5 are kept as they are. If "fail", the function stops with an error if any variant has an MAF greater than 0.5.

sex	if NULL, all samples are treated the same without any modifications; if sex is a factor with levels F (female) and M (male) that is as long as subset or as the VCF file has samples, this argument is interpreted as the sex of the samples. In this case, the genotypes corresponding to male samples are doubled before further processing. This is designed for mixed-sex analyses of the X chromosome outside of the pseudoautosomal regions.
...	for the three latter methods above, all other parameters are passed on to the method with signature TabixFile,GRanges.

Details

This method uses the tabix API provided by the **Rsamtools** package to read from a VCF file, parses the result into a sparse matrix along with positional information, and returns the result as a GenotypeMatrix object. Reading can be restricted to certain regions by specifying the regions object. Note that it might not be possible to read a very large VCF file as a whole.

For all variants, filters in terms of missing values and MAFs can be applied. Moreover, variants with MAFs greater than 0.5 can filtered out or inverted. For details, see descriptions of parameters na.limit, MAF.limit, na.action, and MAF.action above.

Value

returns an object of class GenotypeMatrix

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

https://github.com/samtools/hts-specs

Li, H., Handsaker, B., Wysoker, A., Fenell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and 1000 Genome Project Data Processing Subgroup (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**, 2078-2079. DOI: doi:10.1093/bioinformatics/btp352.

See Also

GenotypeMatrix

Examples

```
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
readGenotypeMatrix(vcfFile)
readGenotypeMatrix(vcfFile, onlyPass=FALSE, MAF.action="ignore")
```

readRegionsFromBedFile

Read Genomic Regions from BED File

Description

Reads a BED file and returns the genomic regions as GRanges object

Usage

```
readRegionsFromBedFile(file, header=FALSE, sep="\t",
                       col.names=c("chrom", "chromStart",
                                   "chromEnd", "names"),
                       ignoreMcols=TRUE, seqInfo=NULL)
```

Arguments

file	the name of the file, text-mode connection, or URL to read data from
header, sep, col.names	arguments passed on to read.table
ignoreMcols	if TRUE (default), further columns are ignored; if FALSE, further columns are appended to the resulting GRanges object as metadata colums (see details below).
seqInfo	can be NULL (default) or an object of class Seqinfo (see details below).

Details

This function is a simple wrapper around the read.table function that reads from a BED file and returns the genomic regions as a GRanges object. How the file is split into columns can be controlled by the arguments header, sep, and col.names. These arguments are passed on to read.table as they are. The choice of the col.names argument is crucial. A wrong col.names argument results in erroneous assignment of columns. The function readRegionsFromBedFile requires columns named "chrom", "chromStart", and "chromEnd" to be present in the object returned from read.table upon reading from the BED file. If a column named "strands" is contained in the BED file, this column is used as strand info in the resulting GRanges object.

If ignoreMcols=TRUE (default), further columns are ignored. If ignoreMcols=FALSE, all columns other than "chrom", "chromStart", "chromEnd", "names", "strand", and "width" are appended to the resulting GRanges object as metadata columns.

Note that the default for col.names has changed in version 1.23.2 of the package. Starting with this version, the BED is no longer assumed to contain strand and width information.

The seqInfo argument can be used to assign the right metadata, such as, genome, chromosome names, and chromosome lengths to the resulting GRanges object.

Value

a GRanges object

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

http://genome.ucsc.edu/FAQ/FAQformat.html#format1

See Also

read.table

Examples

```
## basic example (hg38 regions of HBA1 and HBA2)
bedFile <- system.file("examples/HBA.bed", package="podkat")
readRegionsFromBedFile(bedFile)

## example with enforcing seqinfo
data(hg38Unmasked)
readRegionsFromBedFile(bedFile, seqInfo=seqinfo(hg38Unmasked))

##
## example with regions targeted by Illumina TruSeq Exome Enrichment kit:
## download file "truseq_exome_targeted_regions.hg19.bed.chr.gz" from
## http://support.illumina.com/downloads/truseq_exome_targeted_regions_bed_file.ilmn
## (follow link "TruSeq Exome Targeted Regions BED file"; these regions
##  are based on hg19)
##
## Not run:
readRegionsFromBedFile("truseq_exome_targeted_regions.hg19.bed.chr.gz")

data(hg19Unmasked)
readRegionsFromBedFile("truseq_exome_targeted_regions.hg19.bed.chr.gz",
                       seqInfo=seqinfo(hg19Unmasked))
## End(Not run)
```

---

readSampleNamesFromVcfHeader

Read Sample Names from VCF File Header

Description

Reads the header of a VCF file and returns sample names as character vector

Usage

```
readSampleNamesFromVcfHeader(file, ...)
```

Arguments

file	a TabixFile object or a character string with a file name of the VCF file to read from; if file is a file name, the method internally creates a TabixFile object for this file name.
...	all additional arguments are passed on internally to scanBcfHeader function from the Rsamtools package.

Details

This function is a simple wrapper around the scanBcfHeader function from the Rsamtools package that scans the header of a VCF file and returns the sample names as a character vector.

Value

a character vector with sample names

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

https://github.com/samtools/hts-specs

Li, H., Handsaker, B., Wysoker, A., Fenell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and 1000 Genome Project Data Processing Subgroup (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**, 2078-2079. DOI: doi:10.1093/bioinformatics/btp352.

See Also

scanBcfHeader

Examples

```
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
readSampleNamesFromVcfHeader(vcfFile)
```

---

readVariantInfo-methods

Read information about variants from VCF file

Description

A fast lightweight function that determines information on variants ocurring in a VCF file and returns the result as a VariantInfo object

Usage

```
## S4 method for signature 'TabixFile,GRanges'
readVariantInfo(file, regions, subset,
                noIndels=TRUE, onlyPass=TRUE,
                na.limit=1, MAF.limit=1,
                na.action=c("impute.major", "omit", "fail"),
                MAF.action=c("ignore", "omit","invert", "fail"),
                omitZeroMAF=TRUE, refAlt=FALSE, sex=NULL)
## S4 method for signature 'TabixFile,missing'
readVariantInfo(file, regions, ...)
## S4 method for signature 'character,GRanges'
readVariantInfo(file, regions, ...)
## S4 method for signature 'character,missing'
readVariantInfo(file, regions, ...)
```

Arguments

`file`	a `TabixFile` object or a character string with a file name of the VCF file to read from; if `file` is a file name, the method internally creates a `TabixFile` object for this file name.
`regions`	a `GRanges` object that specifies which genomic regions to read from the VCF file; if missing, the entire VCF file is read.
`subset`	a numeric vector with indices or a character vector with names of samples to restrict to; if specified, only these samples' genotypes are considered when determining the minor allele frequencies (MAFs) of variants.
`noIndels`	if `TRUE` (default), only single-nucleotide variants (SNVs) are considered and indels are skipped.
`onlyPass`	if `TRUE` (default), only variants are considered whose value in the `FILTER` column is "PASS".
`na.limit`	all variants with a missing value ratio above this threshold will be omitted from the output object.
`MAF.limit`	all variants with an MAF above this threshold will be omitted from the output object.
`na.action`	if "impute.major", all missing values are considered as major alleles when computing MAFs. If "omit", all variants containing missing values will be omitted in the output object. If "fail", the function stops with an error if a variant contains any missing values.
`MAF.action`	if "ignore" (default), no action is taken for variants with an MAF greater than 0.5, these variants are kept and included in the output object as they are. If "omit", all variants with an MAF greater than 0.5 are omitted in the output object. If "fail", the function stops with an error if any variant has an MAF greater than 0.5. If "invert", all variants with an MAF exceeding 0.5 will be inverted in the sense that all minor alleles will be replaced by major alleles and vice versa. Note: if this setting is used in conjunction with `refAlt=TRUE`, the MAFs of the variants that have been inverted do no longer correspond to the true alternate allele.

omitZeroMAF	if TRUE (default), variants with an MAF of 0 are not considered and omitted from the output object.
refAlt	if TRUE, two metadata columns named "ref" and "alt" are added to the output object that contain reference and alternate alleles. Note that these sequences can be quite long for indels, which may result in large memory consumption. The default is FALSE.
sex	if NULL, all samples are treated the same without any modifications; if sex is a factor with levels F (female) and M (male) that is as long as subset or as the VCF file has samples, this argument is interpreted as the sex of the samples. In this case, the genotypes corresponding to male samples are doubled before computing MAFs. The option to supply the sex argument is meant to allow for a correct estimate of MAFs as readGenotypeMatrix and assocTest compute it. Note, however, that the MAFs computed in this way do not correspond to the true MAFs contained in the data.
...	for the three latter methods above, all other parameters are passed on to the method with signature TabixFile,GRanges.

Details

This method uses the "tabix" API provided by the **Rsamtools** package to parse a VCF file. The readVariantInfo method considers each variant and determines its minor allele frequency (MAF) and the type of the variant. The result is returned as a VariantInfo object, i.e. a GRanges object with two metadata columns "MAF" and "type". The former contains the MAF of each variant, while the latter is a factor column that contains information about the type of the variant. Possible values in this column are "INDEL" (insertion or deletion), "MULTIPLE" (single-nucleotide variant with multiple alternate alleles), "TRANSITION" (single-nucleotide variation A/G or C/T), "TRANSVERSION" (single-nucleotide variation A/C, A/T, C/G, or G/T), or "UNKNOWN" (anything else). If refAlt is TRUE, two further metadata columns "ref" and "alt" are included which contain reference and alternate alleles of each variant.

For all variants, filters in terms of missing values and MAFs can be applied. Moreover, variants with MAFs greater than 0.5 can filtered out or inverted. For details, see descriptions of parameters na.limit, MAF.limit, na.action, and MAF.action above.

Value

returns an object of class VariantInfo

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

https://github.com/samtools/hts-specs

Li, H., Handsaker, B., Wysoker, A., Fenell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and 1000 Genome Project Data Processing Subgroup (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**, 2078-2079. DOI: doi:10.1093/bioinformatics/btp352.

See Also

GenotypeMatrix

Examples

```
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")

## default parameters
vInfo <- readVariantInfo(vcfFile)
vInfo
summary(vInfo)

## including zero MAF variants and reference/alternate alleles
vInfo <- readVariantInfo(vcfFile, omitZeroMAF=FALSE, refAlt=TRUE)
vInfo
summary(vInfo)
```

---

sort-methods *Sort Association Test Results*

Description

Rearrange association test results according to sort criterion

Usage

```
## S4 method for signature 'AssocTestResultRanges'
sort(x, decreasing=FALSE,
     sortBy=c("p.value", "p.value.adj", "p.value.resampled",
              "p.value.resampled.adj", "genome"))
```

Arguments

x	object of class AssocTestResultRanges
decreasing	logical indicating if sorting should be done in decreasing order
sortBy	sort criterion (see details below)

Details

The function sort takes an AssocTestResultRanges object x and returns a new object of the same class, but with the regions rearranged according to the sort criterion sortBy. As an example, if sortBy is "p.value" regions are sorted according to raw p-values in ascending order, analogously for the choices "p.value.adj", "p.value.resampled", and "p.value.resampled.adj". If sortBy is "genome", the regions are arranged along the genome in the same way as a GRanges object would be sorted. If decreasing is TRUE, the order is reversed.

Value

an AssocTestResultRanges object;

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

AssocTestResultRanges

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)

## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <-read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
nm.lin <- nullModel(y ~ ., pheno)

## perform association test for multiple regions
res <- assocTest(Z, nm.lin, windows)

## perform multiple testing correction
res.adj <- p.adjust(res)

## show sorted results (default: by raw p-value)
as(sort(res), "GRanges")
print(sort(res), sortBy="none")

## show results sorted by adjusted p-value
as(sort(res.adj, sortBy="p.value.adj"), "GRanges")
print(sort(res.adj, sortBy="p.value.adj"), sortBy="none")
```

split-methods *Split* GRanges *Object*

Description

Splits a GRanges object into a GRangesList

Usage

```
## S4 method for signature 'GRanges,GRangesList'
split(x, f)
```

Arguments

x	object of class GRanges
f	object of class GRangesList

Details

This function splits a GRanges object x along a GRangesList object f. More specifically, each region in x is checked for overlaps with every list component of f. The function returns a GRangesList object each component of which contains all overlaps of x with one of the components of f. If the overlap is empty, this component is discarded.

This function is mainly made for splitting regions of interests (transcripts, exons, regions targeted by exome capturing) along chromosomes (and pseudoautosomal regions).

The returned object inherits sequence infos (chromosome names, chromosome lengths, genome, etc.) from the GRangesList object f.

For greater universality, the function takes strand information into account. If overlaps should not be determined in a strand-specific manner, all strand information must be discarded from x and f before calling split.

Value

a GRangesList object (see details above)

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

GRanges, GRangesList

Examples

```
## set up toy example
chr1 <- GRanges(seqnames="chr1", ranges=IRanges(start=1, end=200000))
chr2 <- GRanges(seqnames="chr2", ranges=IRanges(start=1, end=180000))
grL <- GRangesList(list(chr1=chr1, chr2=chr2))
seqlevels(grL) <- c("chr1", "chr2")
seqlengths(grL) <- c(chr1=200000, chr2=180000)
grL

## split set of regions given as 'GRanges' object
gr <- GRanges(seqnames=c("chr1", "chr1", "chr2", "chr2", "chr2"),
              ranges=IRanges(start=c(1, 30000, 10000, 51000, 110000),
                             end=c(340, 37000, 10100, 61000, 176000)))
gr
split(gr, grL)

## consider transcripts on the X chromosome, but with pseudoautosomal
## regions treated separately
if (require(TxDb.Hsapiens.UCSC.hg38.knownGene))
{
    data(hg38Unmasked)

    hg38tr <- transcripts(TxDb.Hsapiens.UCSC.hg38.knownGene)
    strand(hg38tr) <- "*"

    split(hg38tr, hg38Unmasked[c("chrX", "X.PAR1", "X.PAR2", "X.XTR")])
}
```

`unmasked-datasets` *Unmasked Regions of Human Genomes*

Description

Pre-built `GRangesList` objects with unmasked regions of different human genome builds

Usage

```
hg18Unmasked
hg19Unmasked
hg38Unmasked
b36Unmasked
b37Unmasked
```

Format

Each of these is a `GRangesList` object with unmasked regions of different human genome builds, as provided by the packages **BSgenome.Hsapiens.UCSC.hg18.masked**, **BSgenome.Hsapiens.UCSC.hg19.masked**, and **BSgenome.Hsapiens.UCSC.hg38.masked**. The two latter, `b36Unmasked` and `b37Unmasked`,

are variants using chromosome names as the genomes b36 and b37 that are frequently used by the Genome Analysis Toolkit (GATK).

All four data sets comprise all 22 autosomal chromosomes, the two sex chromosomes, mitochondrial DNA, and the six pseudoautosomal regions as defined in the data frames pseudoautosomal.hg18 (for hg18), pseudoautosomal.hg19 (for hg19), and pseudoautosomal.hg38 (for hg38) as provided by the **GWASTools** package. If this is undesired, the user can re-unite the pseudoautosomal regions with their chromosomes as shown in the example section below or run unmaskedRegions him- or herself to extract unmasked regions.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

unmaskedRegions, GRangesList, GRanges, pseudoautosomal

Examples

```
## load data sets
data(hg19Unmasked)
data(b37Unmasked)

## show chromosome names
seqlevels(hg19Unmasked)
seqlevels(b37Unmasked)

## show names of list components
names(hg19Unmasked)
names(b37Unmasked)

## determine numbers of regions for each component
sapply(hg19Unmasked, length)
sapply(b37Unmasked, length)

## re-unite pseudoautosomal regions with their chromosomes
## (that is the result of what unmaskedRegions() would have given
## without specifying pseudoautosomal regions)
hg19basic <- hg19Unmasked[paste0("chr", 1:22)]
hg19basic$chrX <- reduce(unlist(hg19Unmasked[c("chrX", "X.PAR1",
                                               "X.PAR2", "X.XTR")]))
hg19basic$chrY <- reduce(unlist(hg19Unmasked[c("chrY", "Y.PAR1",
                                               "Y.PAR2", "Y.XTR")]))

## show some information about the newly created object
names(hg19basic)
sapply(hg19basic, length)
```

unmaskedRegions *Extract Unmasked Regions from* MaskedBSgenome *Object*

Description

Create a GRangesList of unmasked regions from a MaskedBSgenome object

Usage

```
unmaskedRegions(x, chrs=character(), pseudoautosomal=NULL,
                ignoreGaps=250, activeMasks=active(masks(x[[1]])))
```

Arguments

x	a MaskedBSgenome object
chrs	a character vector of chromosome names to restrict to; if empty (default), all chromosomes in x are considered.
pseudoautosomal	if NULL (default), the chromosomes are considered as they are; pseudoautosomal must be a data frame complying with the format of the pseudoautosomal.hg18, pseudoautosomal.hg19, and pseudoautosomal.hg38 from the **GWASTools** package (see details below).
ignoreGaps	skip assembly gaps only if larger than this threshold; in turn, if two unmasked regions are separated by an assembly gap not larger than ignoreGaps, they are joined in the resuling GRanges object.
activeMasks	masks to apply for determining unmasked region; defaults to the masks that are active by default in the MaskedBSgenome object x. Therefore, this argument only needs to be set if a masking other than the default is necessary.

Details

This function takes a MaskedBSgenome object x and extracts the genomic regions that are unmasked in this genome, where the set of masks to apply can be specified using the activeMasks argument. The result is returned as a GRangesList object each component of which corresponds to one chromosome of the genome x - or a subset thereof if the chrs argument has been specified.

The pseudoautosomal argument allows for a special treatment of pseudoautosomal regions. If not NULL, this argument must be a data frame that contains columns with names "chrom", "start.base", and "end.base". The "chrom" column must contain chromosome names as they appear in the MaskedBSgenome object x. The columns "start.base" and "end.base" must contain numeric values that specify the starts and ends of pseudoautosomal regions, respectively. The function is implemented such that the data frames pseudoautosomal.hg18, pseudoautosomal.hg19, and pseudoautosomal.hg38 provided by the **GWASTools** package can be used (except for the chromosome names that need to be adapted to hg18/hg19/hg38). If the pseudoautosomal argument is specified correctly, the unmaskedRegions function produces separate components in the resulting GRangesList object - one for each pseudoautosomal region. These components are named as the corresponding row names in the data frame pseudoautosomal. Moreover, these regions are omitted from the list of unmasked regions of the chromosomes they are on.

Value

a GRangesList object (see details above)

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

GRangesList, pseudoautosomal

Examples

```
## load packages to obtain masked hg38genome and
##  pseudoautosomal.hg19 from GWASTools package
if (require(BSgenome.Hsapiens.UCSC.hg38.masked) && require(GWASTools))
{
    ## extract unmasked regions of all autosomal chromosomes
    regions <- unmaskedRegions(BSgenome.Hsapiens.UCSC.hg38.masked,
                               chrs=paste0("chr", 1:22))
    names(regions)
    regions$chr1

    ## adjust chromosome names
    pseudoautosomal.hg38
    psaut <- pseudoautosomal.hg38
    psaut$chrom <- paste0("chr", psaut$chrom)
    psaut

    ## extract unmasked regions of sex chromosomes taking pseudoautosomal
    ## regions into account
    regions <- unmaskedRegions(BSgenome.Hsapiens.UCSC.hg38.masked,
                               chrs=c("chrX", "chrY"), pseudoautosomal=psaut)
    names(regions)
    regions$chrX
    regions$X.PAR1

    ## check overlap between X chromosome and a pseudoautosomal region
    intersect(regions$chrX, regions$X.PAR1)
}
```

VariantInfo-class *Class* VariantInfo

Description

S4 class for storing information about variants

Details

This class extends the class GRanges without adding any extra slots. The main difference is that VariantInfo objects always have a metadata column "MAF" that contains minor allele frequencies (MAFs). A special summary method allows for computing statistics about MAFs and types of variants.

Objects of this class should only be created and manipulated by the constructors and accessors described below, as only these methods ensure the integrity of the created objects. Direct modification of object slots is strongly discouraged!

Constructors

variantInfo signature(x="missing"): creates an empty VariantInfo object

variantInfo signature(x="GRanges"): coerces a GRanges object to class VariantInfo by adding a "MAF" metadata column that is initialized with NA values.

Furthermore, see the help page of readVariantInfo.

Accessors

MAF signature(object="VariantInfo"): returns a numeric vector with the minor allele frequencies (MAFs).

Methods

summary signature(object="VariantInfo"): returns a string with the number of variants and metadata columns (if any); if the optional argument details is set to TRUE, this method computes and prints a summary about the MAFs and variant types (if available); this variant returns a list with summarized values.

All other methods, including sub-setting, are inherited from the GRanges class.

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

GRanges, readVariantInfo, genotypeMatrix, readGenotypeMatrix

Examples

```
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")

## read variant info directly from VCF file
vInfo <- readVariantInfo(vcfFile, omitZeroMAF=FALSE)
vInfo
summary(vInfo)

## read full genotype from VCF file
geno <- readGenotypeMatrix(vcfFile)

## display summary of variant info stored in genotype matrix
summary(variantInfo(geno))
```

---

weightFuncs *Weighting Functions*

Description

Functions for computing SNV weights from minor allele frequences (MAF)

Usage

```
betaWeights(x, shape1=1, shape2=25)
logisticWeights(x, th=0.07, slope=150)
invSdWeights(x)
```

Arguments

x	a numeric vector of minor allele frequencies (MAFs); see details below
shape1, shape2	shape parameters of Beta distribution weighting function (see dbeta for details)
th, slope	parameters of the logistic weighting function (see details below)

Details

The function betaWeights is a wrapper around the dbeta function. It uses the same parameters shape1 and shape2, but does not support the non-centrality parameter ncp. The defaults are shape1=1 and shape2=25 as suggested by *Wu et al. (2011)* and implemented in the **SKAT** package. If called without argument x, a function with a single argument x is returned that can directly be used as weighting function, e.g. passed as weightFunc argument to the assocTest method.

The function logisticWeights provides a logistic weighting that corresponds to a soft threshold function. The th parameter corresponds to the threshold and the slope parameter corresponds

to the steepness of the soft threshold. Like `betaWeights`, this function can be called without `x` argument to produce a parameter-free weighting function.

The function `invSdWeights` computes weights as suggested by *Madsen and Browning (2009)*. For consistency, this function also returns a single-argument function if called without `x` argument.

For mathematical details, see Subsection 9.3 of the package vignette.

Value

a numeric vector with weights as long as the argument `x`, a function if `x` was missing;

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

Wu, M. C., Lee, S., Cai, T., Li, Y., Boehnke, M., and Lin, X. (2011) Rare-variant association testing for sequencing data with the sequence kernel association test. *Am. J. Hum. Genet.* **89**, 82-93. DOI: doi:10.1016/j.ajhg.2011.05.029.

Madsen, B. E. and Browning, S. R. (2009) A groupwise association test for rare mutations using a weighted sum statistic. *PLoS Genetics* **5**, e1000384. DOI: doi:10.1371/journal.pgen.1000384

See Also

`GenotypeMatrix`, `dbeta`, `assocTest`

Examples

```
## create a toy example
A <- matrix(rbinom(50, 2, prob=0.2), 5, 10)
MAF <- colSums(A) / (2 * nrow(A))

## compute some weight vectors
betaWeights(MAF, 1, 25)
betaWeights(MAF, 1, 30)
logisticWeights(MAF)
invSdWeights(MAF)

## plot weighting functions (note the missing 'x' arguments)
plot(betaWeights(shape2=30), xlim=c(0, 1))
plot(logisticWeights(), xlim=c(0, 1))
plot(invSdWeights, xlim=c(0, 1))
```

weights *Extract Contribution Weights of Variants*

Description

Method for extracting the contributions that each variant makes to the test statistic of an association test

Usage

```
## S4 method for signature 'AssocTestResult'
weights(object, Z, model)
## S4 method for signature 'AssocTestResultRanges'
weights(object, Z, model, limit=20, sex=NULL)
```

Arguments

object	an object of class AssocTestResult or AssocTestResultRanges
Z	an object of class GenotypeMatrix, an object of class TabixFile, or a character string with the file name of a VCF file
model	an object of class NullModel
limit	maximum number of regions to be processed; set to Inf or non-numeric value like NA or NULL to disable limitation. Do this with caution, in particular, when reading from a VCF file, as reading of excessively large regions from VCF files may take very long or even kill the R session because of excessive memory comsumption!
sex	if NULL, all samples are treated the same without any modifications; if sex is a factor with levels F (female) and M (male) that is as long as the number of samples in model, this argument is interpreted as the sex of the samples. In this case, the genotypes corresponding to male samples are doubled before further processing. This is designed for mixed-sex analyses of the X chromosome outside of the pseudoautosomal regions.

Details

Upon successful completion of an association test, the weights method allows for finding out the individual contributions each of the variants made to the test statistic. This computation is only possible for kernels "linear.podkat" and "linear.SKAT" (see computeKernel).

If called for an AssocTestResult object as first argument object, a GenotypeMatrix object Z, and a NullModel object model, weights returns a GRanges object that contains all variants of variantInfo(Z) along with two numerical metadata columns named "weight.raw" and "weight.contribution". The column "weight.raw" corresponds to raw contributions. These are signed, i.e. a positive value indicates a positive association, while a negative value indicates a negative association. The larger the absolute value, the larger the contribution. The column "weight.contribution" corresponds to relative contributions. These values are non-negative and they sum up to 1. For mathematical details, see Subsection 9.4 of the package vignette.

If weights is called for an AssocTestResultRanges object object, a second argument Z that is an object of class GenotypeMatrix, an object of class TabixFile, or a character string with the name of a VCF file, and a NullModel object model, the contribution weights described above are computed for each region in object. In this case, the method returns a GRangesList with as many components as object has regions, where each list component is a GRanges object containing the contribution weights as described above.

It is essential for weights to work correctly that object is actually the result of an association test between Z and model. If called for objects that actually do not belong to each other, the results are void. The method is implemented such that all possible checks are made that possibly detect inconsistencies between the input objects. However, the final responsibility is left to user to make sure that all data are consistent. Special caution is necessary if weights is run for an AssocTestResultRanges object that has been obtained by merging multiple AssocTestResultRanges using the c method. The c method performs several checks to ensure consistency of association test parameters among the merged results, but the sex parameter is an exception: if it appears to be inconsistent among the results to merge, it is omitted from the merged object (see also AssocTestResultRanges).

The weights method needs to re-evaluate some computations of the association test. In case it is called for Z being a TabixFile object or file name of a VCF file, weights even needs to re-read the genotype data from the file. Therefore, the method has a safety limit not to process too many regions (see limit argument described above).

Value

an object of class GRanges or GRangesList (see details above)

Author(s)

Ulrich Bodenhofer

References

https://github.com/UBod/podkat

See Also

assocTest, AssocTestResult, AssocTestResultRanges, nullModel, NullModel, computeKernel, GenotypeMatrix, p.adjust, filterResult

Examples

```
## load genome description
data(hgA)

## partition genome into overlapping windows
windows <- partitionRegions(hgA)

## load genotype data from VCF file
vcfFile <- system.file("examples/example1.vcf.gz", package="podkat")
Z <- readGenotypeMatrix(vcfFile)
```

```
## read phenotype data from CSV file (continuous trait + covariates)
phenoFile <- system.file("examples/example1lin.csv", package="podkat")
pheno <- read.table(phenoFile, header=TRUE, sep=",")

## train null model with all covariates in data frame 'pheno'
model <- nullModel(y ~ ., pheno)

## perform association test
res <- assocTest(Z, model, windows)

## perform multiple testing correction and filter for
## significant regions
res <- filterResult(p.adjust(res), filterBy="p.value.adj")

## compute contributions
contrib <- weights(res, Z, model)
contrib

## extract most indicative variants
filterResult(contrib)

## plot contributions
plot(contrib[[1]], "weight.raw")
plot(contrib[[1]], "weight.contribution", type="b", alongGenome=TRUE)
```

Index

∗ **classes**
AssocTestResult-class, 10
AssocTestResultRanges-class, 11
GenotypeMatrix-class, 18
NullModel-class, 29
VariantInfo-class, 58
∗ **datasets**
hgA, 24
unmasked-datasets, 54
∗ **methods**
assocTest, 4
filterResult-methods, 16
genotypeMatrix-methods, 20
nullModel, 25
p.adjust-methods, 32
partitionRegions-methods, 33
plot, 35
print-methods, 39
qqplot, 41
readGenotypeMatrix-methods, 43
readVariantInfo-methods, 48
sort-methods, 51
split-methods, 53
weights, 61
∗ **package**
podkat-package, 3
[,GenotypeMatrix,index,index,missing-method *(GenotypeMatrix-class)*, 18
[,GenotypeMatrix,index,missing,missing-method *(GenotypeMatrix-class)*, 18
[,GenotypeMatrix,missing,index,missing-method *(GenotypeMatrix-class)*, 18
[,NullModel,index,missing,missing-method *(NullModel-class)*, 29

assocTest, *3*, 4, *10–13*, *28*, *29*, *34*, *50*, *59*, *60*, *62*
assocTest,character,NullModel-method *(assocTest)*, 4
assocTest,GenotypeMatrix,NullModel-method *(assocTest)*, 4
assocTest,matrix,NullModel-method *(assocTest)*, 4
assocTest,TabixFile,NullModel-method *(assocTest)*, 4
assocTest-methods *(assocTest)*, 4
AssocTestResult, *7–9*, *40*, *61*, *62*
AssocTestResult *(AssocTestResult-class)*, 10
AssocTestResult-class, 10
AssocTestResultRanges, *7–9*, *16*, *17*, *32*, *33*, *36–40*, *42*, *43*, *51*, *52*, *61*, *62*
AssocTestResultRanges *(AssocTestResultRanges-class)*, 11
AssocTestResultRanges-class, 11

b36Unmasked *(unmasked-datasets)*, 54
b37Unmasked *(unmasked-datasets)*, 54
betaWeights *(weightFuncs)*, 59

c,AssocTestResultRanges-method *(AssocTestResultRanges-class)*, 11
class:AssocTestResult *(AssocTestResult-class)*, 10
class:AssocTestResultRanges *(AssocTestResultRanges-class)*, 11
class:GenotypeMatrix *(GenotypeMatrix-class)*, 18
class:NullModel *(NullModel-class)*, 29
class:VariantInfo *(VariantInfo-class)*, 58
coefficients,NullModel-method *(NullModel-class)*, 29
colnames, *18*
computeKernel, *7*, *9*, 14, *61*, *62*

dbeta, *59*, *60*

dgCMatrix, *18, 19*, *21–23*

elementMetadata, *12*
eSet, *21*, *23*

filterResult, *9*, *13*, *62*
filterResult *(*filterResult-methods*)*, 16
filterResult,AssocTestResultRanges-method *(*filterResult-methods*)*, 16
filterResult,GRanges-method *(*filterResult-methods*)*, 16
filterResult,GRangesList-method *(*filterResult-methods*)*, 16
filterResult-methods, 16

GenotypeMatrix, *5*, *7*, *9*, *14, 15*, *19, 20*, *22, 23*, *36–38*, *40*, *43–45*, *51*, *60–62*
GenotypeMatrix *(*GenotypeMatrix-class*)*, 18
genotypeMatrix, *18, 19*, *59*
genotypeMatrix *(*genotypeMatrix-methods*)*, 20
genotypeMatrix,ANY,character,missing-method *(*genotypeMatrix-methods*)*, 20
genotypeMatrix,ANY,GRanges,missing-method *(*genotypeMatrix-methods*)*, 20
genotypeMatrix,ANY,missing,missing-method *(*genotypeMatrix-methods*)*, 20
genotypeMatrix,ANY,numeric,character-method *(*genotypeMatrix-methods*)*, 20
genotypeMatrix,eSet,character,character-method *(*genotypeMatrix-methods*)*, 20
genotypeMatrix,eSet,character,missing-method *(*genotypeMatrix-methods*)*, 20
genotypeMatrix,eSet,numeric,character-method *(*genotypeMatrix-methods*)*, 20
GenotypeMatrix-class, 18
genotypeMatrix-methods, 20
glm, *27–29*
GRanges, *5*, *12, 13*, *16, 17*, *21–25*, *34*, *36–38*, *44*, *46*, *49–51*, *53*, *55, 56*, *58, 59*, *61, 62*
GRangesList, *5, 6*, *16, 17*, *34*, *53–57*, *62*

hg18Unmasked *(*unmasked-datasets*)*, 54
hg19Unmasked *(*unmasked-datasets*)*, 54
hg38Unmasked *(*unmasked-datasets*)*, 54
hgA, 24
hsv, *37*

invSdWeights *(*weightFuncs*)*, 59

length,NullModel-method *(*NullModel-class*)*, 29
lm, *27–29*
logisticWeights *(*weightFuncs*)*, 59

MAF *(*VariantInfo-class*)*, 58
MAF,GenotypeMatrix-method *(*GenotypeMatrix-class*)*, 18
MAF,VariantInfo-method *(*VariantInfo-class*)*, 58
makePSOCKcluster, *6*
MaskedBSgenome, *34*, *56*
Matrix, *14*
mcols, *12*
method:genotypeMatrix *(*genotypeMatrix-methods*)*, 20
method:readGenotypeMatrix *(*readGenotypeMatrix-methods*)*, 43
method:readVariantInfo *(*readVariantInfo-methods*)*, 48

names,NullModel-method *(*NullModel-class*)*, 29
NullModel, *5*, *7–9*, *28, 29*, *40*, *61, 62*
NullModel *(*NullModel-class*)*, 29
nullModel, *3*, *5*, *7–9*, 25, *29–31*, *62*
nullModel,formula,data.frame-method *(*nullModel*)*, 25
nullModel,formula,missing-method *(*nullModel*)*, 25
nullModel,matrix,factor-method *(*nullModel*)*, 25
nullModel,matrix,numeric-method *(*nullModel*)*, 25
nullModel,missing,factor-method *(*nullModel*)*, 25
nullModel,missing,numeric-method *(*nullModel*)*, 25
NullModel-class, 29
nullModel-methods *(*nullModel*)*, 25

p.adjust, *9*, *12*, *17*, *32, 33*, *62*
p.adjust *(*p.adjust-methods*)*, 32
p.adjust,AssocTestResultRanges-method *(*p.adjust-methods*)*, 32
p.adjust-methods, 32

p.adjust.methods, *32*
partitionRegions, *25*
partitionRegions
(partitionRegions-methods), 33
partitionRegions,GRanges-method
(partitionRegions-methods), 33
partitionRegions,GRangesList-method
(partitionRegions-methods), 33
partitionRegions,MaskedBSgenome-method
(partitionRegions-methods), 33
partitionRegions-methods, 33
plot, *9*, *13*, 35, *37*, *42*
plot,AssocTestResultRanges,character-method
(plot), 35
plot,AssocTestResultRanges,GRanges-method
(plot), 35
plot,AssocTestResultRanges,missing-method
(plot), 35
plot,GenotypeMatrix,factor-method
(plot), 35
plot,GenotypeMatrix,missing-method
(plot), 35
plot,GenotypeMatrix,numeric-method
(plot), 35
plot,GRanges,character-method (plot), 35
plot-methods (plot), 35
plot.default, *37*
podkat (podkat-package), 3
podkat-package, 3
points, *36*, *37*
ppoints, *42*
print, *13*
print (print-methods), 39
print,AssocTestResultRanges-method
(print-methods), 39
print-methods, 39
pseudoautosomal, *55*, *57*
pseudoautosomal.hg18, *55*, *56*
pseudoautosomal.hg19, *55*, *56*
pseudoautosomal.hg38, *55*, *56*

qqplot, *9*, *13*, 41, *42*
qqplot,AssocTestResultRanges,AssocTestResultRanges-method
(qqplot), 41
qqplot,AssocTestResultRanges,missing-method
(qqplot), 41
qqplot-methods (qqplot), 41

read.table, *46*, *47*
readGenotypeMatrix, *8*, *9*, *18*, *19*, *50*, *59*
readGenotypeMatrix
(readGenotypeMatrix-methods), 43
readGenotypeMatrix,character,GRanges-method
(readGenotypeMatrix-methods), 43
readGenotypeMatrix,character,missing-method
(readGenotypeMatrix-methods), 43
readGenotypeMatrix,TabixFile,GRanges-method
(readGenotypeMatrix-methods), 43
readGenotypeMatrix,TabixFile,missing-method
(readGenotypeMatrix-methods), 43
readGenotypeMatrix-methods, 43
readRegionsFromBedFile, 46
readSampleNamesFromVcfHeader, 47
readVariantInfo, *58*, *59*
readVariantInfo
(readVariantInfo-methods), 48
readVariantInfo,character,GRanges-method
(readVariantInfo-methods), 48
readVariantInfo,character,missing-method
(readVariantInfo-methods), 48
readVariantInfo,TabixFile,GRanges-method
(readVariantInfo-methods), 48
readVariantInfo,TabixFile,missing-method
(readVariantInfo-methods), 48
readVariantInfo-methods, 48
residuals,NullModel-method
(NullModel-class), 29
rgb, *37*
rownames, *18*
Rsamtools, *48*

scanBcfHeader, *48*
Seqinfo, *46*
show,AssocTestResult-method
(AssocTestResult-class), 10
show,AssocTestResultRanges-method
(AssocTestResultRanges-class), 11
show,GenotypeMatrix-method
(GenotypeMatrix-class), 18
show,NullModel-method
(NullModel-class), 29

show,VariantInfo-method
(VariantInfo-class), 58
SnpSet, *23*
SOCKcluster, *6*, *8*
sort, *13*
sort (sort-methods), 51
sort,AssocTestResultRanges-method
(sort-methods), 51
sort-methods, 51
split (split-methods), 53
split,GRanges,GRangesList-method
(split-methods), 53
split-methods, 53
summary,VariantInfo-method
(VariantInfo-class), 58

TabixFile, *5*, *7*, *44*, *48*, *49*, *61*, *62*
text, *37*

unmasked-datasets, 54
unmaskedRegions, *34*, *55*, 56

VariantInfo, *18*, *19*, *48*, *50*
VariantInfo (VariantInfo-class), 58
variantInfo (VariantInfo-class), 58
variantInfo,GenotypeMatrix-method
(GenotypeMatrix-class), 18
variantInfo,GRanges-method
(VariantInfo-class), 58
variantInfo,missing-method
(VariantInfo-class), 58
VariantInfo-class, 58

weightFuncs, *5*, *9*, 59
weights, 61
weights,AssocTestResult-method
(weights), 61
weights,AssocTestResultRanges-method
(weights), 61
weights-methods (weights), 61