Package 'GWASTools'

April 10, 2015

Version 1.12.2

Type Package

Title Tools for Genome Wide Association Studies

Description Classes for storing very large GWAS data sets and annotation, and functions for GWAS data cleaning and analysis.

Author Stephanie M. Gogarten, Cathy Laurie, Tushar Bhangale, Matthew P. Conomos, Cecelia Laurie, Caitlin McHugh, Ian Painter, Xiuwen Zheng, Jess Shen, Rohit Swarnkar, Adrienne Stilp

Maintainer Stephanie M. Gogarten <sdmorris@u.washington.edu>, Adrienne Stilp <amstilp@u.washington.edu>

Depends Biobase, ncdf, gdsfmt

Imports methods, DBI, RSQLite, GWASExactHW, DNAcopy, survival, sandwich, lmtest, quantsmooth

Suggests GWASdata, BiocGenerics, RUnit, SNPRelate, snpStats, VariantAnnotation

License Artistic-2.0

LazyData yes

biocViews SNP, GeneticVariability, QualityControl, Microarray

Collate utils.R AllGenerics.R AllClasses.R genotypeToCharacter.R Methods-ScanAnnotationDataFrame.R Methods-SnpAnnotationDataFrame.R Methods-ScanAnnotationSQLite.R Methods-SnpAnnotationSQLite.R Methods-NcdfReader.R Methods-NcdfGenotypeReader.R Methods-NcdfIntensityReader.R Methods-GdsReader.R Methods-GdsGenotypeReader.R Methods-GdsIntensityReader.R Methods-MatrixGenotypeReader.R Methods-GenotypeData.R Methods-IntensityData.R createDataFile.R createAffyIntensityFile.R checkGenotypeFile.R checkIntensityFile.R setMissingGenotypes.R imputedDosageFile.R checkImputedDosageFile.R ncdfSubset.R ncdfSubsetCheck.R gdsSubset.R gdsSubsetCheck.R ncdfCreate.R ncdfAddData.R ncdfAddIntensity.R ncdfCheckIntensity.R ncdfCheckGenotype.R ncdfSetMissingGenotypes.R gdsSetMissingGenotypes.R ncdfImputedDosage.R gdsImputedDosage.R gdsCheckImputedDosage.R

plinkUtils.R snpStats.R vcfWrite.R convertNcdfGds.R
convertVcfGds.R BAFfromClusterMeans.R BAFfromGenotypes.R
sdByScanChromWindow.R medianSdOverAutosomes.R
meanSdByChromWindow.R findBAFvariance.R anomSegmentBAF.R
anomFilterBAF.R anomDetectBAF.R LOHfind.R LOHselectAnoms.R
anomDetectLOH.R anomSegStats.R anomStatsPlot.R
anomIdentifyLowQuality.R alleleFrequency.R apartSnpSelection.R
assocTestCPH.R assocTestFisherExact.R assocTestRegression.R
batchChisqTest.R batchFisherTest.R chromIntensityPlot.R
duplicateDiscordance.R duplicateDiscordanceAcrossDatasets.R
duplicateDiscordanceProbability.R genoClusterPlot.R
genoClusterPlotByBatch.R gwasExactHW.R hetByScanChrom.R
hetBySnpSex.R intensityOutliersPlot.R manhattanPlot.R
mendelErr.R meanIntensityByScanChrom.R
missingGenotypeByScanChrom.R missingGenotypeBySnpSex.R
pseudoautoIntensityPlot.R qqPlot.R qualityScoreByScan.R
qualityScoreBySnp.R snpCorrelationPlot.R ibdAreasDraw.R
ibdAssignRelatedness.R ibdPlot.R findRelationsMeanVar.R
pedigreeCheck.R pedigreeDeleteDuplicates.R
pedigreePairwiseRelatedness.R pedigreeMaxUnrelated.R
simulateGenotypeMatrix.R simulateIntensityMatrix.R

R topics documented:

`GWASTools-package` *Tools for Genome Wide Association Studies*

Description

This package contains tools for facilitating cleaning (quality control and quality assurance) and analysis of GWAS data.

Details

GWASTools provides a set of classes for storing data and annotation from Genome Wide Association studies, and a set of functions for data cleaning and analysis that operate on those classes.

Genotype and intensity data are stored in external files (GDS or NetCDF), so it is possible to analyze data sets that are too large to be contained in memory. The `GenotypeReader` class and `IntensityReader` class unions provide a common interface for GDS and NetCDF files.

Two sets of classes for annotation are provided. `SnpAnnotationDataFrame` and `ScanAnnotationDataFrame` extend `AnnotatedDataFrame` and provide in-memory containers for SNP and scan annotation and metadata. `SnpAnnotationSQLite` and `ScanAnnotationSQLite` provide interfaces to SNP and scan annotation and metadata stored in SQLite databases.

The `GenotypeData` and `IntensityData` classes combine genotype or intensity data with SNP and scan annotation, ensuring that the data in the NetCDF files is consistent with annotation through unique SNP and scan IDs. A majority of the functions in the GWASTools package take `GenotypeData` and/or `IntensityData` objects as arguments.

Author(s)

Stephanie M. Gogarten, Cathy Laurie, Tushar Bhangale, Matthew P. Conomos, Cecelia Laurie, Caitlin McHugh, Ian Painter, Xiuwen Zheng, Jess Shen, Rohit Swarnkar, Adrienne Stilp

Maintainer: Stephanie M. Gogarten <sdmorris@u.washington.edu>

References

Laurie, C. C., Doheny, K. F., Mirel, D. B., Pugh, E. W., Bierut, L. J., Bhangale, T., Boehm, F., Caporaso, N. E., Cornelis, M. C., Edenberg, H. J., Gabriel, S. B., Harris, E. L., Hu, F. B., Jacobs, K. B., Kraft, P., Landi, M. T., Lumley, T., Manolio, T. A., McHugh, C., Painter, I., Paschall, J., Rice, J. P., Rice, K. M., Zheng, X., and Weir, B. S., for the GENEVA Investigators (2010), Quality control and quality assurance in genotypic data for genome-wide association studies. Genetic Epidemiology, 34: 591-602. doi: 10.1002/gepi.20516

`alleleFrequency` *Allelic frequency*

Description

Calculates the frequency of the A allele over the specifed scans.

Usage

```
alleleFrequency(genoData, scan.exclude,
                verbose = TRUE)
```

Arguments

`genoData`	`GenotypeData` object.
`scan.exclude`	Integer vector with IDs of scans to exclude.
`verbose`	Logical value specifying whether to show progress information.

Details

Counts male heterozygotes on the X and Y chromosomes as missing values, and any genotype for females on the Y chromosome as missing values. A "sex" variable must be present in the scan annotation slot of `genoData`.

Value

A matrix of allelic frequencies with snps as rows and 4 columns ("M" for males, "F" for females, "all" for all scans, "MAF" for minor allele frequency over all scans).

Author(s)

Cathy Laurie

See Also

`GenotypeData`

Examples

```
library(GWASdata)
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)

# need scan annotation with sex
data(illuminaScanADF)
genoData <-  GenotypeData(gds, scanAnnot=illuminaScanADF)

afreq <- alleleFrequency(genoData, scan.exclude=(illuminaScanADF$race != "CEU"))
close(genoData)
```

`allequal` *Test if two objects have the same elements*

Description

`allequal` tests if two objects have all the same elements, including whether they have `NA`s in the same place.

Usage

```
allequal(x, y)
```

Arguments

x	first object to compare
y	second object to compare

Details

Unlike `all(x == y)`, `allequal` will return `FALSE` if either object is `NULL`. Does not check class types, so `allequal` will return `TRUE` in some cases where `identical` will return `FALSE` (e.g. if two objects are identical when coerced to the same class). `allequal` always retuns a logical value, so it can be used safely in `if` expressions.

Value

Returns `TRUE` if x and y exist and all elements are equal, `FALSE` if some elements are unequal. If there are `NA` values, returns `TRUE` if is.na(x) == is.na(y) and all other elements are equal. Returns `FALSE` if is.na(x) != is.na(y). Retuns `FALSE` if x or y (but not both) is `NULL`.

Author(s)

Stephanie Gogarten

See Also

`identical`, `all`, `all.equal`

Examples

```
x <- c(1,2,NA,4); y <- c(1,2,NA,4);
allequal(x, y) ## TRUE
allequal(1, as.integer(1)) ## TRUE
allequal(1, "1") ## TRUE
```

---

anomDetectBAF *BAF Method for Chromosome Anomaly Detection*

Description

anomSegmentBAF for each sample and chromosome, breaks the chromosome up into segments marked by change points of a metric based on B Allele Frequency (BAF) values.

anomFilterBAF selects segments which are likely to be anomalous.

anomDetectBAF is a wrapper to run anomSegmentBAF and anomFilterBAF in one step.

Usage

```
anomSegmentBAF(intenData, genoData, scan.ids, chrom.ids, snp.ids,
  smooth = 50, min.width = 5, nperm = 10000, alpha = 0.001,
  verbose = TRUE)

anomFilterBAF(intenData, genoData, segments, snp.ids, centromere,
  low.qual.ids = NULL, num.mark.thresh = 15, long.num.mark.thresh = 200,
  sd.reg = 2, sd.long = 1, low.frac.used = 0.1, run.size = 10,
  inter.size = 2, low.frac.used.num.mark = 30, very.low.frac.used = 0.01,
  low.qual.frac.num.mark = 150, lrr.cut = -2, ct.thresh = 10,
  frac.thresh = 0.1, verbose=TRUE,
  small.thresh=2.5, dev.sim.thresh=0.1, centSpan.fac=1.25, centSpan.nmark=50)

anomDetectBAF(intenData, genoData, scan.ids, chrom.ids, snp.ids,
  centromere, low.qual.ids = NULL, ...)
```

Arguments

intenData	An IntensityData object containing the B Allele Frequency. The order of the rows of intenData and the snp annotation are expected to be by chromosome and then by position within chromosome. The scan annotation should contain sex, coded as "M" for male and "F" for female.
genoData	A GenotypeData object. The order of the rows of genoData and the snp annotation are expected to be by chromosome and then by position within chromosome.
scan.ids	vector of scan ids (sample numbers) to process
chrom.ids	vector of (unique) chromosomes to process. Should correspond to integer chromosome codes in intenData. Recommended to include all autosomes, and optionally X (males will be ignored) and the pseudoautosomal (XY) region.

snp.ids	vector of eligible snp ids. Usually exclude failed and intensity-only SNPs. Also recommended to exclude an HLA region on chromosome 6 and XTR region on X chromosome. See HLA and pseudoautosomal. If there are SNPs annotated in the centromere gap, exclude these as well (see centromeres).
smooth	number of markers for smoothing region. See smooth.CNA in the **DNAcopy** package.
min.width	minimum number of markers for a segment. See segment in the **DNAcopy** package.
nperm	number of permutations for deciding significance in segmentation. See segment in the **DNAcopy** package.
alpha	significance level. See segment in the **DNAcopy** package.
verbose	logical indicator whether to print information about the scan id currently being processed. anomSegmentBAF prints each scan id; anomFilterBAF prints a message after every 10 samples: "processing ith scan id out of n" where "ith" with be 10, 10, etc. and "n" is the total number of samples
segments	data.frame of segments from anomSegmentBAF. Names must include "scanID", "chromosome", "num.mark", "left.index", "right.index", "seg.mean". Here "left.index" and "right.index" are row indices of intenData. Left and right refer to start and end of anomaly,respectively, in position order.
centromere	data.frame with centromere position information. Names must include "chrom", "left.base", "right.base". Valid values for "chrom" are 1:22, "X", "Y", "XY". Here "left.base" and "right.base" are base positions of start and end of centromere location in position order. Centromere data tables are provided in centromeres.
low.qual.ids	scan ids determined to be low quality for which some segments are filtered based on more stringent criteria. Default is NULL. Usual choice are scan ids for which median BAF across autosomes > 0.05. See sdByScanChromWindow and medianSdOverAutosomes.
num.mark.thresh	minimum number of SNP markers in a segment to be considered for anomaly
long.num.mark.thresh	min number of markers for "long" segment to be considered for anomaly for which significance threshold criterion is allowed to be less stringent
sd.reg	number of baseline standard deviations of segment mean from a baseline mean for "normal" needed to declare segment anomalous. This number is given by abs(mean of segment - baseline mean)/(baseline standard deviation)
sd.long	same meaning as sd.reg but applied to "long" segments
low.frac.used	if fraction of heterozygous or missing SNP markers compared with number of eligible SNP markers in segment is below this, more stringent criteria are applied to declare them anomalous.
run.size	min length of run of missing or heterozygous SNP markers for possible determination of homozygous deletions
inter.size	number of homozygotes allowed to "interrupt" run for possible determination of homozygous deletions

`low.frac.used.num.mark`
: number of markers threshold for `low.frac.used` segments (which are not declared homozygous deletions

`very.low.frac.used`
: any segments with (num.mark)/(number of markers in interval) less than this are filtered out since they tend to be false positives

`low.qual.frac.num.mark`
: minimum num.mark threshold for low quality scans (`low.qual.ids`) for segments that are also below low.frac.used threshold

`lrr.cut`
: look for runs of LRR values below `lrr.cut` to adjust homozygous deletion endpoints

`ct.thresh`
: minimum number of LRR values below `lrr.cut` needed in order to adjust

`frac.thresh`
: investigate interval for homozygous deletion only if `lrr.cut` and `ct.thresh` thresholds met and (# LRR values below `lrr.cut`)/(# eligible SNPs in segment) > `frac.thresh`

`small.thresh`
: sd.fac threshold use in making merge decisions involving small num.mark segments

`dev.sim.thresh`
: relative error threshold for determining similarity in BAF deviations; used in merge decisions

`centSpan.fac`
: thresholds increased by this factor when considering filtering/keeping together left and right halves of centromere spanning segments

`centSpan.nmark`
: minimum number of markers under which centromere spanning segments are automatically filtered out

`...`
: arguments to pass to `anomFilterBAF`

Details

`anomSegmentBAF` uses the function `segment` from the `DNAcopy` package to perform circular binary segmentation on a metric based on BAF values. The metric for a given sample/chromosome is sqrt(min(BAF,1-BAF,abs(BAF-median(BAF))) where the median is across BAF values on the chromosome. Only BAF values for heterozygous or missing SNPs are used.

`anomFilterBAF` determines anomalous segments based on a combination of thresholds for number of SNP markers in the segment and on deviation from a "normal" baseline. (See `num.mark.thresh`,`long.num.mark.thresh`, `sd.reg`, and `sd.long`.) The "normal" baseline metric mean and standard deviation are found across all autosomes not segmented by `anomSegmentBAF`. This is why it is recommended to include all autosomes for the argument `chrom.ids` to ensure a more accurate baseline.

Some initial filtering is done, including possible merging of consecutive segments meeting `sd.reg` threshold along with other criteria (such as not spanning the centromere) and adjustment for accurate break points for possible homozygous deletions (see `lrr.cut`, `ct.thresh`, `frac.thresh`, `run.size`, and `inter.size`). Male samples for X chromosome are not processed.

More stringent criteria are applied to some segments (see `low.frac.used`,`low.frac.used.num.mark`, `very.low.frac.used`, `low.qual.ids`, and `low.qual.frac.num.mark`).

`anomDetectBAF` runs `anomSegmentBAF` with default values and then runs `anomFilterBAF`. Additional parameters for `anomFilterBAF` may be passed as arguments.

Value

anomSegmentBAF returns a data.frame with the following elements: Left and right refer to start and end of anomaly, respectively, in position order.

scanID	integer id of scan
chromosome	chromosome as integer code
left.index	row index of intenData indicating left endpoint of segment
right.index	row index of intenData indicating right endpoint of segment
num.mark	number of heterozygous or missing SNPs in the segment
seg.mean	mean of the BAF metric over the segment

anomFilterBAF and anomDetectBAF return a list with the following elements:

raw — data.frame of raw segmentation data, with same output as anomSegmentBAF as well as:

- left.base: base position of left endpoint of segment
- right.base: base position of right endpoint of segment
- sex: sex of scan.id coded as "M" or "F"
- sd.fac: measure of deviation from baseline equal to abs(mean of segment - baseline mean)/(baseline standard deviation); used in determining anomalous segments

filtered — data.frame of the segments identified as anomalies, with the same columns as raw as well as:

- merge: TRUE if segment was a result of merging. Consecutive segments from output of anomSegmentBAF that meet certain criteria are merged.
- homodel.adjust: TRUE if original segment was adjusted to narrow in on a homozygous deletion
- frac.used: fraction of (eligible) heterozygous or missing SNP markers compared with total number of eligible SNP markers in segment

base.info — data frame with columns:

- scanID: integer id of scan
- base.mean: mean of non-anomalous baseline. This is the mean of the BAF metric for heterozygous and missing SNPs over all unsegmented autosomes that were considered.
- base.sd: standard deviation of non-anomalous baseline
- chr.ct: number of unsegmented chromosomes used in determining the non-anomalous baseline

seg.info — data frame with columns:

- scanID: integer id of scan
- chromosome: chromosome as integer
- num.segs: number of segments produced by anomSegmentBAF

Note

It is recommended to include all autosomes as input. This ensures a more accurate determination of baseline information.

Author(s)

Cecelia Laurie

References

See references in segment in the package **DNAcopy**. The BAF metric used is modified from Itsara,A., *et.al* (2009) Population Analysis of Large Copy Number Variants and Hotspots of Human Genetic Disease. *American Journal of Human Genetics*, **84**, 148–161.

See Also

segment and smooth.CNA in the package **DNAcopy**, also findBAFvariance, anomDetectLOH

Examples

```
library(GWASdata)
data(illuminaScanADF, illuminaSnpADF)

blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
bl <- GdsIntensityReader(blfile)
blData <-  IntensityData(bl, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <-  GenotypeData(geno, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

# segment BAF
scan.ids <- illuminaScanADF$scanID[1:2]
chrom.ids <- unique(illuminaSnpADF$chromosome)
snp.ids <- illuminaSnpADF$snpID[illuminaSnpADF$missing.n1 < 1]
seg <- anomSegmentBAF(blData, genoData, scan.ids=scan.ids,
                      chrom.ids=chrom.ids, snp.ids=snp.ids)

# filter segments to detect anomalies
data(centromeres.hg18)
filt <- anomFilterBAF(blData, genoData, segments=seg, snp.ids=snp.ids,
                      centromere=centromeres.hg18)

# alternatively, run both steps at once
anom <- anomDetectBAF(blData, genoData, scan.ids=scan.ids, chrom.ids=chrom.ids,
                      snp.ids=snp.ids, centromere=centromeres.hg18)

close(blData)
close(genoData)
```

anomDetectLOH *LOH Method for Chromosome Anomaly Detection*

Description

anomDetectLOH breaks a chromosome up into segments of homozygous runs of SNP markers determined by change points in Log R Ratio and selects segments which are likely to be anomalous.

Usage

```
anomDetectLOH(intenData, genoData, scan.ids, chrom.ids, snp.ids,
  known.anoms, smooth = 50, min.width = 5, nperm = 10000, alpha = 0.001,
  run.size = 50, inter.size = 4, homodel.min.num = 10, homodel.thresh = 10,
  small.num = 20, small.thresh = 2.25, medium.num = 50, medium.thresh = 2,
  long.num = 100, long.thresh = 1.5, small.na.thresh = 2.5,
  length.factor = 5, merge.fac = 0.85, min.lrr.num = 20, verbose = TRUE)
```

Arguments

intenData	An IntensityData object containing the Log R Ratio. The order of the rows of intenData and the snp annotation are expected to be by chromosome and then by position within chromosome. The scan annotation should contain sex, coded as "M" for male and "F" for female.
genoData	A GenotypeData object. The order of the rows of genoData and the snp annotation are expected to be by chromosome and then by position within chromosome.
scan.ids	vector of scan ids (sample numbers) to process
chrom.ids	vector of (unique) chromosomes to process. Should correspond to integer chromosome codes in intenData. Recommended for use with autosomes, X (males will be ignored), and the pseudoautosomal (XY) region.
snp.ids	vector of eligible snp ids. Usually exclude failed and intensity-only snps. Also recommended to exclude an HLA region on chromosome 6 and XTR region on X chromosome. See HLA and pseudoautosomal. If there are SNPs annotated in the centromere gap, exclude these as well (see centromeres).
known.anoms	data.frame of known anomalies (usually from anomDetectBAF); must have "scanID","chromosome","left.i Here "left.index" and "right.index" are row indices of intenData. Left and right refer to start and end of anomaly, respectively, in position order.
smooth	number of markers for smoothing region. See smooth.CNA in the **DNAcopy** package.
min.width	minimum number of markers for segmenting. See segment in the **DNAcopy** package.
nperm	number of permutations. See segment in the **DNAcopy** package.
alpha	significance level. See segment in the **DNAcopy** package.

run.size	number of markers to declare a 'homozygous' run (here 'homozygous' includes homozygous and missing)
inter.size	number of consecutive heterozygous markers allowed to interrupt a 'homozygous' run
homodel.min.num	minimum number of markers to detect extreme difference in lrr (for homozygous deletion)
homodel.thresh	threshold for measure of deviation from non-anomalous needed to declare segment a homozygous deletion.
small.num	minimum number of SNP markers to declare segment as an anomaly (other than homozygous deletion)
small.thresh	threshold for measure of deviation from non-anomalous to declare segment anomalous if number of SNP markers is between small.num and medium.num.
medium.num	threshold for number of SNP markers to identify 'medium' size segment
medium.thresh	threshold for measure of deviation from non-anomalous needed to declare segment anomalous if number of SNP markers is between medium.num and long.num.
long.num	threshold for number of SNP markers to identify 'long' size segment
long.thresh	threshold for measure of deviation from non-anomalous when number of markers is bigger than long.num
small.na.thresh	threshold measure of deviation from non-anomalous when number of markers is between small.num and medium.num and 'local mad.fac' is NA. See Details section for definition of 'local mad.fac'.
length.factor	window around anomaly defined as length.factor*(no. of markers in segment) on either side of the given segment. Used in determining 'local mad.fac'. See Details section.
merge.fac	threshold for 'sd.fac'= number of baseline standard deviations of segment mean from baseline mean; consecutive segments with 'sd.fac' above threshold are merged
min.lrr.num	if any 'non-anomalous' interval has fewer markers than min.lrr.num, interval is ignored in finding non-anomalous baseline unless it's the only piece left
verbose	logical indicator whether to print the scan id currently being processed

Details

Detection of anomalies with loss of heterozygosity accompanied by change in Log R Ratio. Male samples for X chromosome are not processed.

Circular binary segmentation (CBS) (using the R-package **DNAcopy**) is applied to LRR values and, in parallel, runs of homozygous or missing genotypes of a certain minimal size (run.size) (and allowing for some interruptions by no more than inter.size heterozygous SNPs) are identified. Intervals from known.anoms are excluded from the identification of runs. After some possible merging of consecutive CBS segments (based on satisfying a threshold merge.fac for deviation from non-anomalous baseline), the homozygous runs are intersected with the segments from CBS.

Determination of anomalous segments is based on a combination of number-of-marker thresholds and deviation from a non-anomalous baseline. Segments are declared anomalous if deviation from non-anomalous is above corresponding thresholds. (See small.num, small.thresh, medium.num,medium.thresh, long.num,long.thresh,and small.na.thresh.) Non-anomalous median and MAD are defined for each sample-chromosome combination. Intervals from known.anoms and the homozygous runs identified are excluded; remaining regions are the non-anomalous baseline.

Deviation from non-anomalous is measured by a combination of a chromosome-wide 'mad.fac' and a 'local mad.fac' (both the average and the minimum of these two measures are used). Here 'mad.fac' is (segment median-non-anomalous median)/(non-anomalous MAD) and 'local mad.fac' is the same definition except the non-anomalous median and MAD are computed over a window including the segment (see length.factor). Median and MADare found for eligible LRR values.

Value

A list with the following elements:

raw — raw homozygous run data, not including any regions present in known.anoms. A data.frame with the following columns: Left and right refer to start and end of anomaly, respectively, in position order.

- left.index: row index of intenData indicating left endpoint of segment
- right.index: row index of intenData indicating right endpoint of segment
- left.base: base position of left endpoint of segment
- right.base: base position of right endpoint of segment
- scanID: integer id of scan
- chromosome: chromosome as integer code

raw.adjusted — data.frame of runs after merging and intersecting with CBS segments, with the following columns: Left and right refer to start and end of anomaly, respectively, in position order.

- scanID: integer id of scan
- chromosome: chromosome as integer code
- left.index: row index of intenData indicating left endpoint of segment
- right.index: row index of intenData indicating right endpoint of segment
- left.base: base position of left endpoint of segment
- right.base: base position of right endpoint of segment
- num.mark: number of eligible SNP markers in segment
- seg.median: median of eligible LRR values in segment
- seg.mean: mean of eligible LRR values in segment
- mad.fac: measure of deviation from non-anomalous baseline, equal to abs(median of segment - baseline median)/(baseline MAD); used in determining anomalous segments
- sd.fac: measure of deviation from non-anomalous baseline, equal to abs(mean of segment - baseline mean)/(baseline standard deviation); used in determining whether to merge

- local: measure of deviation from non-anomalous baseline used equal to abs(median of segment - local baseline median)/(local baseline MAD); local baseline consists of eligible LRR values in a window around segment; used in determining anomalous segments
- num.segs: number of segments found by CBS for the given chromosome
- chrom.nonanom.mad: MAD of eligible LRR values in non-anomalous regions across the chromosome
- chrom.nonanom.median: median of eligible LRR values in non-anomalous regions across the chromosome
- chrom.nonanom.mean: mean of eligible LRR values in non-anomalous regions across the chromosome
- chrom.nonanom.sd: standard deviation of eligible LRR values in non-anomalous regions across the chromosome
- sex: sex of the scan id coded as "M" or "F"

filtered — data.frame of the segments identified as anomalies. Columns are the same as in raw.adjusted.

base.info — data.frame with columns:

- chrom.nonanom.mad: MAD of eligible LRR values in non-anomalous regions across the chromosome
- chrom.nonanom.median: median of eligible LRR values in non-anomalous regions across the chromosome
- chrom.nonanom.mean: mean of eligible LRR values in non-anomalous regions across the chromosome
- chrom.nonanom.sd: standard deviation of eligible LRR values in non-anomalous regions across the chromosome
- sex: sex of the scan id coded as "M" or "F"
- num.runs: number of original homozygous runs found for given scan/chromosome
- num.segs: number of segments for given scan/chromosome produced by CBS
- scanID: integer id of scan
- chromosome: chromosome as integer code
- sex: sex of the scan id coded as "M" or "F"

segments — data.frame of the segmentation found by CBS with columns:

- scanID: integer id of scan
- chromosome: chromosome as integer code
- left.index: row index of intenData indicating left endpoint of segment
- right.index: row index of intenData indicating right endpoint of segment
- left.base: base position of left endpoint of segment
- right.base: base position of right endpoint of segment
- num.mark: number of eligible SNP markers in the segment
- seg.mean: mean of eligible LRR values in the segment
- sd.fac: measure of deviation from baseline equal to abs(mean of segment - baseline mean)/(baseline standard deviation) where the baseline is over non-anomalous regions

merge — data.frame of scan id/chromosome pairs for which merging occurred.

- scanID: integer id of scan
- chromosome: chromosome as integer code

Author(s)

Cecelia Laurie

References

See references in segment in the package **DNAcopy**.

See Also

segment and smooth.CNA in the package **DNAcopy**, also findBAFvariance, anomDetectLOH

Examples

```
library(GWASdata)
data(illuminaScanADF, illuminaSnpADF)

blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
bl <- GdsIntensityReader(blfile)
blData <-  IntensityData(bl, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <-  GenotypeData(geno, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

scan.ids <- illuminaScanADF$scanID[1:2]
chrom.ids <- unique(illuminaSnpADF$chromosome)
snp.ids <- illuminaSnpADF$snpID[illuminaSnpADF$missing.n1 < 1]

# example for known.anoms, would get this from anomDetectBAF
known.anoms <- data.frame("scanID"=scan.ids[1],"chromosome"=21,
  "left.index"=100,"right.index"=200)

LOH.anom <- anomDetectLOH(blData, genoData, scan.ids=scan.ids,
  chrom.ids=chrom.ids, snp.ids=snp.ids, known.anoms=known.anoms)

close(blData)
close(genoData)
```

---

anomIdentifyLowQuality

Identify low quality samples

Description

Identify low quality samples for which false positive rate for anomaly detection is likely to be high. Measures of noise (high variance) and high segmentation are used.

Usage

```
anomIdentifyLowQuality(snp.annot, med.sd, seg.info,
  sd.thresh, sng.seg.thresh, auto.seg.thresh)
```

Arguments

snp.annot	SnpAnnotationDataFrame with column "eligible", where "eligible" is a logical vector indicating whether a SNP is eligible for consideration in anomaly detection (usually FALSE for HLA and XTR regions, failed SNPs, and intensity-only SNPs). See HLA and pseudoautosomal.
med.sd	data.frame of median standard deviation of BAlleleFrequency (BAF) or LogR-Ratio (LRR) values across autosomes for each scan, with columns "scanID" and "med.sd". Usually the result of medianSdOverAutosomes. Usually only eligible SNPs are used in these computations. In addition, for BAF, homozygous SNPS are excluded.
seg.info	data.frame with segmentation information from anomDetectBAF or anomDetectLOH. Columns must include "scanID", "chromosome", and "num.segs". (For anomDetectBAF, segmentation information is found in $seg.info from output. For anomDetectLOH, segmentation information is found in $base.info from output.)
sd.thresh	Threshold for med.sd above which scan is identified as low quality. Suggested values are 0.1 for BAF and 0.25 for LOH.
sng.seg.thresh	Threshold for segmentation factor for a given chromosome, above which the chromosome is said to be highly segmented. See Details. Suggested values are 0.0008 for BAF and 0.0048 for LOH.
auto.seg.thresh	Threshold for segmentation factor across autosome, above which the scan is said to be highly segmented. See Details. Suggested values are 0.0001 for BAF and 0.0006 for LOH.

Details

Low quality samples are determined separately with regard to each of the two methods of segmentation, anomDetectBAF and anomDetectLOH. BAF anomalies (respectively LOH anomalies) found for samples identified as low quality for BAF (respectively LOH) tend to have a high false positive rate.

A scan is identified as low quality due to high variance (noise), i.e. if med.sd is above a certain threshold sd.thresh.

High segmentation is often an indication of artifactual patterns in the B Allele Frequency (BAF) or Log R Ratio values (LRR) that are not always captured by high variance. Here segmentation information is determined by anomDetectBAF or anomDetectLOH which use circular binary segmentation implemented by the R-package **DNAcopy**. The measure for high segmentation is a "segmentation factor" = (number of segments)/(number of eligible SNPS). A single chromosome

segmentation factor uses information for one chromosome. A segmentation factor across autosomes uses the total number of segments and eligible SNPs across all autosomes. See `med.sd`, `sd.thresh`, `sng.seg.thresh`, and `auto.seg.thresh`.

Value

A data.frame with the following columns:

`scanID`	integer id for the scan
`chrX.num.segs`	number of segments for chromosome X
`chrX.fac`	segmentation factor for chromosome X
`max.autosome`	autosome with highest single segmentation factor
`max.auto.fac`	segmentation factor for chromosome = `max.autosome`
`max.auto.num.segs`	number of segments for chromosome = `max.autosome`
`num.ch.segd`	number of chromosomes segmented, i.e. for which change points were found
`fac.all.auto`	segmentation factor across all autosomes
`med.sd`	median standard deviation of BAF (or LRR values) across autosomes. See `med.sd` in Arguments section.
`type`	one of the following, indicating reason for identification as low quality:

- `auto.seg`: segmentation factor `fac.all.auto` above `auto.seg.thresh` but `med.sd` acceptable
- `sd`: standard deviation factor `med.sd` above `sd.thresh` but `fac.all.auto` acceptable
- `both.sd.seg`: both high variance and high segmentation factors, `fac.all.auto` and `med.sd`, are above respective thresholds
- `sng.seg`: segmentation factor `max.auto.fac` is above `sng.seg.thresh` but other measures acceptable
- `sng.seg.X`: segmentation factor `chrX.fac` is above `sng.seg.thresh` but other measures acceptable

Author(s)

Cecelia Laurie

See Also

`findBAFvariance`, `anomDetectBAF`, `anomDetectLOH`

Examples

```
library(GWASdata)
data(illuminaScanADF, illuminaSnpADF)

blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
bl <- GdsIntensityReader(blfile)
blData <-  IntensityData(bl, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)
```

```
genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <-  GenotypeData(geno, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

# initial scan for low quality with median SD
baf.sd <- sdByScanChromWindow(blData, genoData)
med.baf.sd <- medianSdOverAutosomes(baf.sd)
low.qual.ids <- med.baf.sd$scanID[med.baf.sd$med.sd > 0.05]

# segment and filter BAF
scan.ids <- illuminaScanADF$scanID[1:2]
chrom.ids <- unique(illuminaSnpADF$chromosome)
snp.ids <- illuminaSnpADF$snpID[illuminaSnpADF$missing.n1 < 1]
data(centromeres.hg18)
anom <- anomDetectBAF(blData, genoData, scan.ids=scan.ids, chrom.ids=chrom.ids,
  snp.ids=snp.ids, centromere=centromeres.hg18, low.qual.ids=low.qual.ids)

# further screen for low quality scans
snp.annot <- illuminaSnpADF
snp.annot$eligible <- snp.annot$missing.n1 < 1
low.qual <- anomIdentifyLowQuality(snp.annot, med.baf.sd, anom$seg.info,
  sd.thresh=0.1, sng.seg.thresh=0.0008, auto.seg.thresh=0.0001)

close(blData)
close(genoData)
```

---

anomSegStats *Calculate LRR and BAF statistics for anomalous segments*

Description

Calculate LRR and BAF statistics for anomalous segments and plot results

Usage

```
anomSegStats(intenData, genoData, snp.ids, anom, centromere,
  lrr.cut = -2, verbose = TRUE)

anomStatsPlot(intenData, genoData, anom.stats, snp.ineligible,
  plot.ineligible = FALSE, centromere = NULL,
  brackets = c("none", "bases", "markers"), brkpt.pct = 10,
  whole.chrom = FALSE, win = 5, win.calc = FALSE, win.fixed = 1,
  zoom = c("both", "left", "right"), main = NULL, info = NULL,
  ideogram = TRUE, ideo.zoom = FALSE, ideo.rect = TRUE,
  mult.anom = FALSE, cex = 0.5, cex.leg = 1.5, ...)
```

Arguments

intenData	An IntensityData object containing BAlleleFreq and LogRRatio. The order of the rows of intenData and the snp annotation are expected to be by chromosome and then by position within chromosome.
genoData	A GenotypeData object. The order of the rows of intenData and the snp annotation are expected to be by chromosome and then by position within chromosome.
snp.ids	vector of eligible SNP ids. Usually exclude failed and intensity-only SNPS. Also recommended to exclude an HLA region on chromosome 6 and XTR region on X chromosome. See HLA and pseudoautosomal. If there are SNPs annotated in the centromere gap, exclude these as well (see centromeres). x
anom	data.frame of detected chromosome anomalies. Names must include "scanID", "chromosome", "left.index", "right.index", "sex", "method", "anom.id". Valid values for "method" are "BAF" or "LOH" referring to whether the anomaly was detected by BAF method (anomDetectBAF) or by LOH method (anomDetectLOH). Here "left.index" and "right.index" are row indices of intenData with left.index < right.index.
centromere	data.frame with centromere position info. Names must include "chrom", "left.base", "right.base". Valid values for "chrom" are 1:22, "X", "Y", "XY". Here "left.base" and "right.base" are start and end base positions of the centromere location, respectively. Centromere data tables are provided in centromeres.
lrr.cut	count the number of eligible LRR values less than lrr.cut
verbose	whether to print the scan id currently being processed
anom.stats	data.frame of chromosome anomalies with statistics, usually the output of anomSegStats. Names must include "anom.id", "scanID", "chromosome", "left.index", "right.index", "method", "nmark.all", "nmark.elig", "left.base", "right.base", "nbase", "non.anom.baf.med", "non.anom.lrr.med", "anom.baf.dev.med", "anom.baf.dev.5", "anom.lrr.med", "nmark.baf", "nmark.lrr". Left and right refer to start and end, respectively, of the anomaly, in position order.
snp.ineligible	vector of ineligible snp ids (e.g., intensity-only, failed snps, XTR and HLA regions). See HLA and pseudoautosomal.
plot.ineligible	whether or not to include ineligible points in the plot for LogRRatio
brackets	type of brackets to plot around breakpoints - none, use base length, use number of markers (note that using markers give asymmetric brackets); could be used, along with brkpt.pct, to assess general accuracy of end points of the anomaly
brkpt.pct	percent of anomaly length in bases (or number of markers) for width of brackets
whole.chrom	logical to plot the whole chromosome or not (overrides win and zoom)
win	size of the window (a multiple of anomaly length) surrounding the anomaly to plot
win.calc	logical to calculate window size from anomaly length; overrides win and gives window of fixed length given by win.fixed
win.fixed	number of megabases for window size when win.calc=TRUE

zoom	indicates whether plot includes the whole anomaly ("both") or zooms on just the left or right breakpoint; "both" is default
main	Vector of titles for upper (LRR) plots. If NULL, titles will include anom.id, scanID, sex, chromosome, and detection method.
info	character vector of extra information to include in the main title of the upper (LRR) plot
ideogram	logical for whether to plot a chromosome ideogram under the BAF and LRR plots.
ideo.zoom	logical for whether to zoom in on the ideogram to match the range of the BAF/LRR plots
ideo.rect	logical for whether to draw a rectangle on the ideogram indicating the range of the BAF/LRR plots
mult.anom	logical for whether to plot multiple anomalies from the same scan-chromosome pair on a single plot. If FALSE (default), each anomaly is shown on a separate plot.
cex	cex value for points on the plots
cex.leg	cex value for the ideogram legend
...	Other parameters to be passed directly to plot.

Details

anomSegStats computes various statistics of the input anomalies. Some of these are basic statistics for the characteristics of the anomaly and for measuring deviation of LRR or BAF from expected. Other statistics are used in downstrean quality control analysis, including detecting terminal anomalies and investigating centromere-spanning anomalies.

anomStatsPlot produces separate png images of each anomaly in anom.stats. Each image consists of an upper plot of LogRRatio values and a lower plot of BAlleleFrequency values for a zoomed region around the anomaly or whole chromosome (depending up parameter choices). Each plot has vertical lines demarcating the anomaly and horizontal lines displaying certain statistics from anomSegStats. The upper plot title includes sample number and chromosome. Further plot annotation describes which anomaly statistics are represented.

Value

anomSegStats produces a data.frame with the variables for anom plus the following columns: Left and right refer to position order with left < right.

nmark.all	total number of SNP markers on the array from left.index to right.index inclusive
nmark.elig	total number of eligible SNP markers on the array from left.index to right.index, inclusive. See snp.ids for definition of eligible SNP markers.
left.base	base position corresponding to left.index
right.base	base position corresponding to right.index
nbase	number of bases from left.index to right.index, inclusive

non.anom.baf.med
: BAF median of non-anomalous segments on all autosomes for the associated sample, using eligible heterozygous or missing SNP markers

non.anom.lrr.med
: LRR median of non-anomalous segments on all autosomes for the associated sample, using eligible SNP markers

non.anom.lrr.mad
: MAD for LRR of non-anomalous segments on all autosomes for the associated sample, using eligible SNP markers

anom.baf.dev.med
: BAF median of deviations from non.anom.baf.med of points used to detect anomaly (eligible and heterozygous or missing)

anom.baf.dev.5
: median of BAF deviations from 0.5, using eligible heterozygous or missing SNP markers in anomaly

anom.baf.dev.mean
: mean of BAF deviations from non.anom.baf.med, using eligible heterozygous or missing SNP markers in anomaly

anom.baf.sd
: standard deviation of BAF deviations from non.anom.baf.med, using eligible heterozygous or missing SNP markers in anomaly

anom.baf.mad
: MAD of BAF deviations from non.anom.baf.med, using eligible heterozygous or missing SNP markers in anomaly

anom.lrr.med
: LRR median of eligible SNP markers within the anomaly

anom.lrr.sd
: standard deviation of LRR for eligible SNP markers within the anomaly

anom.lrr.mad
: MAD of LRR for eligible SNP markers within the anomaly

nmark.baf
: number of SNP markers within the anomaly eligible for BAF detection (eligible markers that are heterozygous or missing)

nmark.lrr
: number of SNP markers within the anomaly eligible for LOH detection (eligible markers)

cent.rel
: position relative to centromere - left, right, span

left.most
: T/F for whether the anomaly is the left-most anomaly for this sample-chromosome, i.e. no other anomalies with smaller start base position

right.most
: T/F whether the anomaly is the right-most anomaly for this sample-chromosome, i.e. no other anomalies with larger end base position

left.last.elig
: T/F for whether the anomaly contains the last eligible SNP marker going to the left (decreasing position)

right.last.elig
: T/F for whether the anomaly contains the last eligible SNP marker going to the right (increasing position)

left.term.lrr.med
: median of LRR for all eligible SNP markers from left-most eligible marker to the left telomere (only calculated for the most distal anom)

right.term.lrr.med
: median of LRR for all eligible markers from right-most eligible marker to the right telomere (only calculated for the most distal anom)

`left.term.lrr.n`
: sample size for calculating `left.term.lrr.med`

`right.term.lrr.n`
: sample size for calculating `right.term.lrr.med`

`cent.span.left.elig.n`
: number of eligible markers on the left side of centromere-spanning anomalies

`cent.span.right.elig.n`
: number of eligible markers on the right side of centromere-spanning anomalies

`cent.span.left.bases`
: length of anomaly (in bases) covered by eligible markers on the left side of the centromere

`cent.span.right.bases`
: length of anomaly (in bases) covered by eligible markers on the right side of the centromere

`cent.span.left.index`
: index of eligible marker left-adjacent to centromere; recall that index refers to row indices of `intenData`

`cent.span.right.index`
: index of elig marker right-adjacent to centromere

`bafmetric.anom.mean`
: mean of BAF-metric values within anomaly, using eligible heterozygous or missing SNP markers BAF-metric values were used in the detection of anomalies. See `anomDetectBAF` for definition of BAF-metric

`bafmetric.non.anom.mean`
: mean of BAF-metric values within non-anomalous segments across all autosomes for the associated sample, using eligible heterozygous or missing SNP markers

`bafmetric.non.anom.sd`
: standard deviation of BAF-metric values within non-anomalous segments across all autosomes for the associated sample, using eligible heterozygous or missing SNP markers

`nmark.lrr.low`
: number of eligible markers within anomaly with LRR values less than `lrr.cut`

Note

The non-anomalous statistics are computed over all autosomes for the sample associated with an anomaly. Therefore the accuracy of these statistics relies on the input anomaly data.frame including all autosomal anomalies for a given sample.

Author(s)

Cathy Laurie

See Also

`anomDetectBAF`, `anomDetectLOH`

Examples

```
library(GWASdata)
data(illuminaScanADF, illuminaSnpADF)

blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
bl <- GdsIntensityReader(blfile)
blData <-  IntensityData(bl, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <-  GenotypeData(geno, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

scan.ids <- illuminaScanADF$scanID[1:2]
chrom.ids <- unique(illuminaSnpADF$chromosome)
snp.ids <- illuminaSnpADF$snpID[illuminaSnpADF$missing.n1 < 1]
snp.failed <- illuminaSnpADF$snpID[illuminaSnpADF$missing.n1 == 1]

# example results from anomDetectBAF
baf.anoms <- data.frame("scanID"=rep(scan.ids[1],2), "chromosome"=rep(21,2),
  "left.index"=c(100,300), "right.index"=c(200,400), sex=rep("M",2),
  method=rep("BAF",2), anom.id=1:2, stringsAsFactors=FALSE)

# example results from anomDetectLOH
loh.anoms <- data.frame("scanID"=scan.ids[2],"chromosome"=22,
  "left.index"=400,"right.index"=500, sex="F", method="LOH",
  anom.id=3, stringsAsFactors=FALSE)

anoms <- rbind(baf.anoms, loh.anoms)
data(centromeres.hg18)
stats <- anomSegStats(blData, genoData, snp.ids=snp.ids, anom=anoms,
  centromere=centromeres.hg18)

anomStatsPlot(blData, genoData, anom.stats=stats,
  snp.ineligible=snp.failed, centromere=centromeres.hg18)

close(blData)
close(genoData)
```

---

apartSnpSelection *Random selection of SNPs*

Description

Randomly selects SNPs for which each pair is at least as far apart as the specified basepair distance.

Usage

```
apartSnpSelection(chromosome, position, min.dist = 1e+05,
                  init.sel = NULL,  max.n.chromosomes = -1,
                  verbose = TRUE)
```

Arguments

chromosome	An integer vector containing the chromosome for each SNP. Valid values are 1-26, any other value will be interpreted as missing and not selected.
position	A numeric vector of the positions (in basepairs) of the SNPs.
min.dist	A numeric value to specify minimum distance required (in basepairs).
init.sel	A logical vector indicating the initial SNPs to be included.
max.n.chromosomes	A numeric value specifying the maximum number of SNPs to return per chromosome, "-1" means no number limit.
verbose	A logical value specifying whether to show progress information while running.

Details

apartSnpSelection selects SNPs randomly with the condition that they are at least as far apart as min.dist in basepairs. The starting set of SNPs can be specified with init.sel.

Value

A logical vector indicating which SNPs were selected.

Author(s)

Xiuwen Zheng

Examples

```
library(GWASdata)
data(affy_snp_annot)
pool <- affy_snp_annot$chromosome < 23
rsnp <- apartSnpSelection(affy_snp_annot$chromosome, affy_snp_annot$position,
                          min.dist=15000, init.sel=pool)
```

---

asSnpMatrix *Utilities for snpStats*

Description

asSnpMatrix converts a GenotypeData object to a SnpMatrix-class object.

Usage

```
asSnpMatrix(genoData, snpNames="snpID", scanNames="scanID",
            snp=c(1,-1), scan=c(1,-1))
```

Arguments

genoData	A GenotypeData object.
snpNames	The name of the SNP variable in genoData to use as the column (SNP) names in the SnpMatrix-class object.
scanNames	The name of the scan variable in genoData to use as the row (scan) names in the SnpMatrix-class object.
snp	An integer vector of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that all SNPs should be read.
scan	An integer vector of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that all scans should be read.

Details

The default is to extract all SNPs and scans from genoData, but for a large dataset this may exceed R's memory limit. Alternatively, snp and scan may be used to specify (start, count) of SNPs and scans to extract from genoData.

In the SnpMatrix object, genotypes are stored as 0 = missing, 1 = "A/A", 2= "A/B" or "B/A", and 3 = "B/B". (In a GenotypeData object, 0 = "B/B", 1 = "A/B" or "B/A", and 2 = "A/A".) Columns are SNPs with names snpNames and rows are scans with names scanNames (the transpose of the GenotypeData object).

Value

A SnpMatrix-class object.

Author(s)

Stephanie Gogarten

See Also

SnpMatrix-class, GenotypeData

Examples

```
library(snpStats)
library(GWASdata)
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)
data(illuminaSnpADF, illuminaScanADF)
genoData <-  GenotypeData(gds, snpAnnot=illuminaSnpADF, scanAnnot=illuminaScanADF)
snpmat <- asSnpMatrix(genoData, snpNames="rsID", scanNames="scanID")
snpmat
as(snpmat[1:5, 1:5], "character")
summary(snpmat)
```

```
# only chromosome 21
chr <- getChromosome(genoData)
c21 <- which(chr == 21)
snpmat <- asSnpMatrix(genoData, snpNames="rsID", scanNames="scanID",
                      snp=c(c21[1], length(c21)))
snpmat
close(genoData)
```

---

assocTestCPH *Cox proportional hazards*

Description

Fits Cox proportional hazards model

Usage

```
assocTestCPH(genoData, event, time.to.event,
             covars, factor.covars = NULL,
             scan.chromosome.filter = NULL,
             scan.exclude = NULL,
             maf.filter = FALSE,
             GxE = NULL, strata.vars = NULL,
             chromosome.set = NULL, block.size = 5000,
             verbose = TRUE,
             outfile = NULL)
```

Arguments

genoData	GenotypeData object, should contain sex and phenotypes in scan annotation. Chromosomes are expected to be in contiguous blocks.
event	name of scan variable in genoData for event to analyze
time.to.event	name of scan variable in genoData for time to event
covars	vector of covariate terms for model (can include interactions as 'a:b', main effects correspond to scan variable names in genoData)
factor.covars	vector of names of covariates to be converted to factor
scan.chromosome.filter	a logical matrix that can be used to exclude some chromosomes, some scans, or some specific scan-chromosome pairs. Entries should be TRUE if that scan-chromosome pair should be included in the analysis, FALSE if not. The number of rows must be equal to the number of scans in genoData, and the number of columns must be equal to the largest integer chromosome value in genoData. The column number must match the chromosome number. e.g. A scan.chromosome.filter matrix used for an analyis when genoData has SNPs with chromosome=(1-24, 26, 27) (i.e. no Y (25) chromosome SNPs) must have 27 columns (all FALSE in the 25th column). But a scan.chromosome.filter matrix used for an analysis genoData has SNPs chromosome=(1-26) (i.e no Unmapped (27) chromosome SNPs) must have only 26 columns.

scan.exclude	an integer vector containing the IDs of entire scans to be excluded.
maf.filter	whether to filter results returned using MAF*(1-MAF) > 75/(2*n) where MAF = minor allele frequency and n = number of events
GxE	name of the covariate to use for E if genotype-by-environment (i.e. SNP:E) model is to be analyzed, in addition to the main effects (E can be a covariate interaction)
strata.vars	vector of names of variables to stratify on for a stratified analysis (use NULL if no stratified analysis needed)
chromosome.set	integer vector with chromosome(s) to be analyzed. Use 23, 24, 25, 26, 27 for X, XY, Y, M, Unmapped respectively.
block.size	number of SNPs from a given chromosome to read in one block from genoData
verbose	Logical value specifying whether to show progress information.
outfile	a character string to append in front of ".chr.i_k.RData" for naming the output data-frames; where i is the first chromosome, and k is the last chromosome used in that call to the function. "chr.i_k." will be omitted if chromosome.set=NULL.

Details

This function performs Cox proportional hazards regression of a survival object (using the Surv function) on SNP genotype and other covariates. It uses the coxph function from the R survival library.

Individual samples can be included or excluded from the analysis using the scan.exclude parameter. Individual chromosomes can be included or excluded by specifying the indices of the chromosomes to be included in the chromosome.set parameter. Specific chromosomes for specific samples can be included or excluded using the scan.chromosome.filter parameter.

Both scan.chromosome.filter and scan.exclude may be used together. If a scan is excluded in EITHER, then it will be excluded from the analysis, but it does NOT need to be excluded in both. This design allows for easy filtering of anomalous scan-chromosome pairs using the scan.chromosome.filter matrix, but still allows easy exclusion of a specific group of scans (e.g. males or Caucasians) using scan.exclude.

The argument maf.filter indicates whether to filter results returned using 2 * MAF * (1-MAF) * n > 75 where MAF = minor allele frequency and n = number of events. This filter was suggested by Ken Rice and Thomas Lumley, who found that without this requirement, at threshold levels of significance for genome-wide studies, Cox regression p-values based on standard asymptotic approximations can be notably anti-conservative.

Value

If outfile=NULL (default), all results are returned as a data.frame. If outfile is specified, no data is returned but the function saves a data.frame with the naming convention as described by the argument outfile. Columns for the main effects model are:

index	snp index
snpID	unique integer ID for SNP
chr	chromosome

maf	minor allele frequency calculated as appropriate for autosomal loci
mafx	minor allele frequency calculated as appropriate for X-linked loci
beta	regression coefficient returned by the coxph function
se	standard error of the regression coefficient returned by the coxph function
z	z statistic returned by the coxph function
pval	p-value for the z-statistic returned by the coxph function
warned	TRUE if a warning was issued
n.events	number of events in complete cases for the given SNP

If GxE is not NULL, another data.frame is returned with the results of the genotype-by-environment model. If outfile=NULL, the function returns a list with names (main, GxE); otherwise the GxE data.frame is saved as a separate output file. Columns are:

index	snp index
snpID	unique integer ID for SNP
chr	chromosome
maf	minor allele frequency calculated as appropriate for autosomal loci
mafx	minor allele frequency calculated as appropriate for X-linked loci
warned	TRUE if a warning was issued
n.events	number of events in complete cases for the given SNP
ge.lrtest	Likelihood ratio test statistic for the GxE interaction
ge.pval	p-value for the likelihood ratio test statistic

Warnings:

If outfile is not NULL, another file will be saved with the name "outfile.chr.i_k.warnings.RData" that contains any warnings generated by the function.

Author(s)

Cathy Laurie

See Also

GenotypeData, coxph

Examples

```
# an example of a scan chromosome matrix
# desiged to eliminate duplicated individuals
# and scans with missing values of sex
library(GWASdata)
data(illuminaScanADF)
scanAnnot <- illuminaScanADF
samp.chr.matrix <- matrix(TRUE,nrow(scanAnnot),26)
dup <- duplicated(scanAnnot$subjectID)
samp.chr.matrix[dup | is.na(scanAnnot$sex),] <- FALSE
```

```
samp.chr.matrix[scanAnnot$sex=="F", 25] <- FALSE

# additionally, exclude YRI subjects
scan.exclude <- scanAnnot$scanID[scanAnnot$race == "YRI"]

# create some variables for the scans
scanAnnot$age <- rnorm(nrow(scanAnnot),mean=40, sd=10)
scanAnnot$event <- rbinom(nrow(scanAnnot),1,0.4)
scanAnnot$ttoe <- rnorm(nrow(scanAnnot),mean=100,sd=10)

# create data object
gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(gdsfile)
genoData <-  GenotypeData(gds, scanAnnot=scanAnnot)

# variables
event <- "event"
time.to.event <- "ttoe"
covars <- c("sex", "age")
factor.covars <- "sex"

chr.set <- 21

res <- assocTestCPH(genoData,
  event="event", time.to.event="ttoe",
  covars=c("sex", "age"), factor.covars="sex",
  scan.chromosome.filter=samp.chr.matrix,
  scan.exclude=scan.exclude,
  chromosome.set=chr.set)

close(genoData)
```

`assocTestFisherExact` *Association tests*

Description

This function performs Fisher's Exact Test using allele counts for cases and controls. It takes the output from `assocTestRegression` as its input.

Usage

```
assocTestFisherExact(dat, outfile = NULL)
```

Arguments

`dat`	a data.frame of output from `assocTestRegression` run with `model.type = "logistic"` (a case/control test). It should contain all columns of the output and only the rows (SNPs) that the user wishes to perform Fisher's Exact Test on.

`outfile` a character string to append in front of "FisherExact.Rdata" for naming the output data-frame. If set to `NULL` (default), then the results are returned to the R console.

Details

This function performs a basic Fisher's Exact Test to test for differences in allele frequencies between cases and controls; it compares the "A" and "B" allele frequencies between cases and controls.

This function uses the output from `assocTestRegression` run with `model.type = "logistic"` as its input. It uses the output genotype counts for cases and controls, converts them to allele counts and performs the Fisher's Exact Test to calculate an allelic odds ratio (the odds of being a case for the minor allele compared to the major allele), a 95% confidence interval, and a p-value.

One suggested use of this function is to perform significance tests on SNPs that are monomorphic in either cases or controls, as a standard logistic regression test is not well-defined in this case. The `assocTestRegression` function will return an error for these SNPs; see its help page for more detail.

Value

If `outfile=NULL` (default), all results are returned as a data.frame. If `outfile` is specified, no data is returned but the function saves a data-frame with the naming convention as described by the variable `outfile`.

The first five columns of the data-frame are taken from `dat`:

`snpID` snpID of the SNP

`n` sample size for the regression

`MAF` minor allele frequency. Note that calculation of allele frequency for the X chromosome is different than that for the autosomes and the XY (pseudo-autosomal) region.

`minor.allele` the minor allele. Takes values "A" or "B".

`regression.warningOrError` report of different possible warnings or errors from the regression test: 0 if controls are monomorphic (logistic regression only), 1 if cases are monomorphic (logistic refression only), 2 if all samples are monomorphic, 9 if a warning or error occured during model fitting, `NA` if none

`Fisher.OR` odds ratio from the Fisher's Exact test of allele counts. It is the odds of being a case for the minor allele compared to the major allele.

`Fisher.OR_L95` lower 95% confidence limit for the odds ratio.

`Fisher.OR_U95` upper 95% confidence limit for the odds ratio.

`Fisher.pval` Fisher's Exact test p-value.

`nA.cc0` number of A alleles among samples with outcome coded as 0

`nB.cc0` number of B alleles among samples with outcome coded as 0

`nA.cc1` number of A alleles among samples with outcome coded as 1

`nB.cc1` number of B alleles among samples with outcome coded as 1

Author(s)

Matthew P. Conomos

See Also

assocTestRegression

Examples

```
# The following example would take the output from association tests run on chromosome 22 using assocTestRegression
# and perform the Fishers Exact Test on those that were monomorphic in either the cases or the controls.
# The output would be saved as "chr22test.FisherExact.RData"

# run assocTestRegression
library(GWASdata)
data(illuminaScanADF)
scanAnnot <- illuminaScanADF

gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(gdsfile)
genoData <-  GenotypeData(gds, scanAnnot=scanAnnot)

mydat <- assocTestRegression(genoData, outcome="status",
  model.type="logistic", chromosome.set=22)

# subset rows of those SNPs that are monomorphic in cases or controls; keep all columns
mono.dat <- mydat[which(mydat$model.1.additive.warningOrError == 0 |
                        mydat$model.1.additive.warningOrError ==1),]

# perform the Fishers Exact Test
assocTestFisherExact(dat = mono.dat, outfile = "chr22test")

# load the output
outfile <- "chr22test.FisherExact.RData"
fisher.res <- getobj(outfile)
head(fisher.res)
unlink(outfile)
close(genoData)
```

---

assocTestRegression *Association tests*

Description

This function performs regression based and likelihood ratio based association tests for both genotype main effects as well as interaction effects. It also computes genotype counts for association tests.

Usage

```
assocTestRegression(genoData, outcome, model.type,
                    covar.list = NULL, ivar.list = NULL,
                    gene.action.list = NULL, dosage = FALSE,
                    scan.chromosome.filter = NULL,
                    scan.exclude = NULL, CI = 0.95,
                    robust = FALSE, LRtest = TRUE,
                    chromosome.set = NULL, block.set = NULL,
                    block.size = 5000, verbose = TRUE,
                    outfile = NULL)
```

Arguments

genoData	GenotypeData object, should contain phenotypes and covariates in scan annotation. Chromosomes are expected to be in contiguous blocks.
outcome	Vector (of length equal to the number of models) of names of the outcome variables for each model. These names must be in the scan annotation of genoData. e.g. c("case.cntl.status", "blood.pressure") will use "case.cntl.status" as the outcome for the first model and "blood pressure" for the second. Outcome variables must be coded as 0/1 for logistic regression.
model.type	vector (of length equal to the number of models) with the types of models to be fitted. The elements should be one of: "logistic", "linear", or "poisson". e.g. c("logistic", "linear") will perform two tests: the first using logistic regression, and the second using linear regression.
covar.list	list (of length equal to the number of models) of vectors containing the names of covariates to be used in the regression model (blank, i.e. "" if none). The default value is NULL and will include no covariates in any of the models. The covariate names must be in the scan annotation of genoData. e.g. `covar.list() <- list(); covar.list[[1]] <- c("age","sex"); covar.list[[2]] <- c("");` will use both "age" and "sex" as covariates for the first model and no covariates for the second model (this regresses on only the genotype).
ivar.list	list (of length equal to the number of models) of vectors containing the names of covariates for which to include an interaction with genotype (blank, i.e. "" if none). The default value is NULL and will include no interactions in any of the models. The covariate names must be in the scan annotation of genoData. e.g. `ivar.list() <- list(); ivar.list[[1]] <- c("sex"); ivar.list[[2]] <- c("");` will include a genotype*"sex" interaction term for the first model and no interactions for the second model.
gene.action.list	a list (of length equal to the number of models) of vectors containing the types of gene action models to be used in the corresponding regression model. Valid options are "additive", "dominant", and "recessive", referring to how the minor allele is treated, as well as "dominance". "additive" coding sets the marker variable for homozygous minor allele samples = 2, heterozygous samples = 1, and homozygous major allele samples = 0. "dominant" coding sets the marker variable for homozygous minor allele samples = 2, heterozygous samples = 2, and homozygous major allele samples = 0. "recessive" coding sets the marker

variable for homozygous minor allele samples = 2, heterozygous samples = 0, and homozygous major allele samples = 0. "dominance" coding sets the marker variable for homozygous minor allele samples = major allele frequency, heterozygous samples = 0, and homozygous major allele samples = minor allele frequency. This coding eliminates the additive component of variance for the marker variable, leaving only the dominance component of variance. The default value is NULL, which assumes only an "additive" gene action model for every test. e.g. `gene.action.list() <- list(); gene.action.list[[1]] <- c("additive"); gene.action.list[[2]] <- c("dominant", "recessive");` will run the first model using "additive" gene action, and will run the second model using both "dominant" and "recessive" gene actions.

`dosage` logical for whether or not the genotype values are imputed dosages. The default value is FALSE for true genotype calls. When using imputed dosages, the `gene.action` must be additive, and genotype counts will not be calculated.

`scan.chromosome.filter`
a logical matrix that can be used to exclude some chromosomes, some scans, or some specific scan-chromosome pairs. Entries should be TRUE if that scan-chromosome pair should be included in the analysis, FALSE if not. The number of rows must be equal to the number of scans in `genoData`, and the number of columns must be equal to the largest integer chromosome value in `genoData`. The column number must match the chromosome number. e.g. A scan.chromosome.filter matrix used for an analyis when `genoData` has SNPs with chromosome=(1-24, 26, 27) (i.e. no Y (25) chromosome SNPs) must have 27 columns (all FALSE in the 25th column). But a scan.chromosome.filter matrix used for an analysis `genoData` has SNPs chromosome=(1-26) (i.e no Unmapped (27) chromosome SNPs) must have only 26 columns.

`scan.exclude` an integer vector containing the IDs of entire scans to be excluded.

`CI` sets the confidence level for the confidence interval calculations. Confidence intervals are computed at every SNP; for the odds ratio when using logistic regression, for the linear trend parameter when using linear regression, and for the rate ratio when using Poisson regression. The default value is 0.95 (i.e. a 95% confidence interval). The confidence level must be between 0 and 1.

`robust` logical for whether to use sandwich-based robust standard errors. The default value is FALSE, and uses model based standard errors. The standard error estimates are returned and also used for Wald Tests of significance.

`LRtest` logical for whether to perform Likelihood Ratio Tests. The default value is TRUE, and performs LR tests in addition to Wald tests (which are always performed). NOTE: Performing the LR tests adds a noticeable amount of computation time.

`chromosome.set` integer vector with chromosome(s) to be analyzed. Use 23, 24, 25, 26, 27 for X, XY, Y, M, Unmapped respectively.

`block.set` list (of length equal to `length(chromosome.set)`) of vectors where every vectors contains the indices of the SNP blocks (on that chromosome) to be analyzed. e.g. `chromosome.set <- c(1,2); block.set <- list(); chr.1 <- c(1,2,3); chr.2 <- c(5,6,7,8); block.set$chr.1 <- chr.1; block.set$chr.2 <- chr.2;` will analyze first three block on chromosome 1 and 5th through 8th blocks on chromosome 2. The actual number of SNPs analyzed will depend on `block.size`.

	Default value is NULL. If block.set == NULL, all the SNPs on chromosomes in chromosome.set will be analyzed.
block.size	Number of SNPs to be read from genoData at once.
verbose	if TRUE (default), will print status updates while the function runs. e.g. it will print "chr 1 block 1 of 10" etc. in the R console after each block of SNPs is done being analyzed.
outfile	a character string to append in front of ".model.j.gene_action.chr.i_k.RData" for naming the output data-frames; where j is the model number, gene_action is the gene.action type, i is the first chromosome, and k is the last chromosome used in that call to the function. "chr.i_k." will be omitted if chromosome.set=NULL. If set to NULL (default), then the results are returned to the R console.

Details

When using models without interaction terms, the association tests compare the model including the covariates and genotype value to the model including only the covariates (a test of genotype effect). When using a model with interaction terms, tests are performed for each of the interaction terms separately as well as a joint test of all the genotype terms (main effects and interactions) to detect any genotype effect. All tests and p-values are always computed using Wald tests with p-values computed from Chi-Squared distribtuions. The option of using either sandwich based robust standard errors (which make no model assumptions) or using model based standard errors for the confidence intervals and Wald tests is specified by the robust parameter. The option of also performing equivalent Likelihood Ratio tests is available and is specified by the LRtest parameter.

Three types of regression models are available: "logistic", "linear", or "poisson". Multiple models can be run at the same time by putting multiple arguments in the outcome, model.type, covar.list, ivar.list, and gene.action.list parameters. For each model, available gene action models are "additive", "dominant", "recessive", and "dominance." See above for the correct usage of each of these.

For logistic regression models, if the SNP is monomorphic in either cases or controls, then the slope parameter is not well-defined. In this situation, an error message will be returned (see model.N.gene_action.warningOrError in the Value section below for details), and the regression of this SNP will not be performed. If a test of significance is still desired for these SNPs, we suggest performing either a Fisher's Exact Test using the assocTestFisherExact function provided in GWASTools or performing a trend test (using model.type = "linear" in this function).

Individual samples can be included or excluded from the analysis using the scan.exclude parameter. Individual chromosomes can be included or excluded by specifying the indices of the chromosomes to be included in the chromosome.set parameter. Specific chromosomes for specific samples can be included or excluded using the scan.chromosome.filter parameter. The inclusion or exclusion of specific blocks of SNP's on each chromosome can be specified using the block.set parameter. Note that the actual SNP's included or excluded will change according to the value of block.size.

Both scan.chromosome.filter and scan.exclude may be used together. If a scan is excluded in EITHER, then it will be excluded from the analysis, but it does NOT need to be excluded in both. This design allows for easy filtering of anomalous scan-chromosome pairs using the scan.chromosome.filter matrix, but still allows easy exclusion of a specific group of scans (e.g. males or Caucasians) using scan.exclude.

This function allows for the usage of imputed dosages in place of genotypes in the additive model by specifying `dosage = TRUE`.

Value

If `outfile=NULL` (default), all results are returned as a single data.frame. If `outfile` is specified, no data is returned but the function saves a data-frame for each model gene-action pair, with the naming convention as described by the variable `outfile`.

The first column of each data-frame is:

`snpID` snpID (from `genoData`) of the SNP

After this first column, for every model gene-action pair there are the following columns: Here, "model.M" is the name assigned to the test where M = 1, 2,..., length(model.type), and "gene_action" is the gene-action type of the test (one of "additive", "dominant", "recessive", or "dominance").

`model.M.n` sample size for the regression

For tests that use linear regression (will be `NA` if using imputed dosages for genotypes):

`model.M.nAA` number of AA genotypes in samples

`model.M.nAB` number of AB genotypes in samples

`model.M.nBB` number of BB genotypes in samples

For tests that use logistic regression (will be `NA` if using imputed dosages for genotypes):

`model.M.nAA.cc0`
number of AA genotypes in samples with outcome coded as 0

`model.M.nAB.cc0`
number of AB genotypes in samples with outcome coded as 0

`model.M.nBB.cc0`
number of BB genotypes in samples with outcome coded as 0

`model.M.nAA.cc1`
number of AA genotypes in samples with outcome coded as 1

`model.M.nAB.cc1`
number of AB genotypes in samples with outcome coded as 1

`model.M.nBB.cc1`
number of BB genotypes in samples with outcome coded as 1

`model.M.MAF` minor allele frequency. Note that calculation of allele frequency for the X chromosome is different than that for the autosomes and the XY (pseudo-autosomal) region. Hence if chromosome.set includes 23, `genoData` should provide the sex of the scan ("M" or "F") i.e. there should be a column named "sex" with "F" for females and "M" for males.

`model.M.minor.allele`
the minor allele. Takes values "A" or "B".

`model.M.gene_action.warningOrError`
report of different possible warnings or errors: 0 if controls are monomorphic (logistic regression only), 1 if cases are monomorphic (logistic refression only), 2 if all samples are monomorphic or allele frequency is NA, 9 if a warning or error occured during model fitting, `NA` if none

`model.M.gene_action.Est.G`
: estimate of the regression coefficient for the genotype term. See the description in `gene.action.list` above for interpretation.

`model.M.gene_action.SE.G`
: standard error of the regression coefficient estimate for the genotype term. Could be either sandwich based (robust) or model based; see description in `robust`.

For tests that use linear regression:

`model.M.gene_action.L95.G`
: lower 95% confidence limit for the genotype coefficient (95 will be replaced with whatever confidence level is chosen in `CI`).

`model.M.gene_action.U95.G`
: upper 95% confidence limit for the genotype coefficient (95 will be replaced with whatever confidence level is chosen in `CI`).

For tests that use logistic regression:

`model.M.gene_action.OR.G`
: odds ratio for the genotype term. This is exp(the regression coefficient). See the description in "gene.action.list" above for interpretation.

`model.M.gene_action.OR_L95.G`
: lower 95% confidence limit for the odds ratio (95 will be replaced with whatever confidence level is chosen in `CI`).

`model.M.gene_action.OR_U95.G`
: upper 95% confidence limit for the odds ratio (95 will be replaced with whatever confidence level is chosen in `CI`).

For tests that use Poisson regression:

`model.M.gene_action.RR.G`
: relative risk for the genotype term. This is exp(the regression coefficient). See the description in "gene.action.list" above for interpretation.

`model.M.gene_action.RR_L95.G`
: lower 95% confidence limit for the relative risk (95 will be replaced with whatever confidence level is chosen in `CI`).

`model.M.gene_action.RR_U95.G`
: upper 95% confidence limit for the relative risk (95 will be replaced with whatever confidence level is chosen in `CI`).

For all regression models:

`model.M.gene_action.Wald.Stat.G`
: value of the Wald test statistic for testing the genotype parameter

`model.M.gene_action.Wald.pval.G`
: Wald test p-value, calculated from a Chi-Squared distribution. This can be calculated using either sandwich based robust standard errors or model based standard errors (see `robust`).

If `LRtest = TRUE`, for tests with no interaction variables:

`model.M.gene_action.LR.Stat.G`
: value of the Likelihood Ratio test statistic for testing the genotype parameter

`model.M.gene_action.LR.pval.G`
: Likelihood Ratio test p-value.

For tests with interaction variables: Here, "ivar_name" refers to the name of the interaction variable; if there are multiple interaction variables, there will be a set of the following columns for each one.

`model.M.gene_action.Est.G:ivar_name`
: estimate of the regression coefficient for the interaction between genotype and ivar_name.

`model.M.gene_action.SE.G:ivar_name`
: standard error of the interaction regression coefficient estimate. Could be either sandwich based (robust) or model based; see description in `robust`.

For tests that use linear regression and interaction variables:

`model.M.gene_action.L95.G:ivar_name`
: lower 95% confidence limit for the genotype*ivar_name interaction coefficient (95 will be replaced with whatever confidence level is chosen in `CI`).

`model.M.gene_action.U95.G:ivar_name`
: upper 95% confidence limit for the genotype*ivar_name interaction coefficient (95 will be replaced with whatever confidence level is chosen in `CI`).

For tests that use logistic regression and interaction variables:

`model.M.gene_action.OR.G:ivar_name`
: odds ratio for the genotype*ivar_name interaction term. This is exp(the interaction regression coefficient). A separate odds ratio is calculated for each interaction term. See the description in "gene.action.list" above for interpretation.

`model.M.gene_action.OR_L95.G:ivar_name`
: lower 95% confidence limit for the odds ratio (95 will be replaced with whatever confidence level is chosen in `CI`).

`model.M.gene_action.OR_U95.G:ivar_name`
: upper 95% confidence limit for the odds ratio (95 will be replaced with whatever confidence level is chosen in `CI`).

For tests that use Poisson regression and interaction variables:

`model.M.gene_action.RR.G:ivar_name`
: relative risk for the genotype*ivar_name interaction term. This is exp(the interaction regression coefficient). A separate relative risk is calculated for each interaction term. See the description in "gene.action.list" above for interpretation.

`model.M.gene_action.RR_L95.G:ivar_name`
: lower 95% confidence limit for the relative risk (95 will be replaced with whatever confidence level is chosen in `CI`).

`model.M.gene_action.RR_U95.G:ivar_name`
: upper 95% confidence limit for the relative risk (95 will be replaced with whatever confidence level is chosen in `CI`).

For all regression models with interaction variables:

`model.M.gene_action.Wald.Stat.G:ivar_name`
: value of the Wald test statistic for testing the genotype*ivar_name interaction parameter

`model.M.gene_action.Wald.pval.G:ivar_name`
: Wald test p-value for testing the genotype*ivar_name interaction parameter, calculated from a Chi-Squared distribution. This can be calculated using either sandwich based robust standard errors or model based standard errors (see `robust`).

If `LRtest = TRUE`, for tests with interaction variables:

`model.M.gene_action.LR.Stat.G:ivar_name`
: value of the Likelihood Ratio test statistic for testing the genotype*ivar_name interaction parameter

`model.M.gene_action.LR.pval.G:ivar_name`
: Likelihood Ratio test p-value for testing the genotype*ivar_name interaction parameter.

For all regression models with interaction variables:

`model.M.gene_action.Wald.Stat.G.Joint`
: value of the Wald test statistic for jointly testing all of the genotype parameters (main effects and interactions); a test for any genotype effect.

`model.M.gene_action.Wald.pval.G.Joint`
: Wald test p-value for jointly testing all of the genotype parameters, calculated from a Chi-Squared distribution. This can be calculated using either sandwich based robust standard errors or model based standard errors (see `robust`).

If `LRtest = TRUE`, for tests with interaction variables:

`model.M.gene_action.LR.Stat.G.Joint`
: value of the Likelihood Ratio test statistic for jointly testing all of the genotype parameters (main effects and interactions); a test for any genotype effect.

`model.M.gene_action.LR.pval.G.Joint`
: Likelihood Ratio test p-value for jointly testing all of the genotype parameters.

Attributes:

There is also an attribute for each output data-frame called "model" that shows the model used for the test. This can be viewed with the following R command: `attr(mod.res, "model")` where mod.res is the output data-frame from the function. The `attr()` command will return something like: model.1.additive "case.cntl.status ~ genotype + age + sex , logistic regression, additive gene action"

There is another attribute called "SE" that shows if Robust or Model Based standard errors were used for the test. This can be viewed with the following R command: `attr(mod.res, "SE")` where mod.res is the output data-frame from the function.

Warnings:

Another file will be saved with the name "outfile.chr.i_k.warnings.RData" that contains any warnings generated by the function. An example of what would be contained in this file: Warning

messages: 1: Model 1 , Y chromosome tests are confounded with sex and should be run separately without sex in the model 2: Model 2 , Y chromosome tests are confounded with sex and should be run separately without sex in the model

Author(s)

Matthew P. Conomos, Tushar Bhangale

See Also

GenotypeData, lm, glm, vcov, vcovHC, lrtest

Examples

```
# The following example would perform 3 tests (from 2 models):
# the first a logistic regression of case.cntl.status on genotype, age, and sex, including an interaction term betwe
# the second a linear regression of blood pressure on genotype using dominant gene action,
# and the third, a linear regression of blood pressure on genotype again, but this time using recessive gene action.
# This test would only use chromosome 21.
# It would perform both robust Wald tests using sandwich based robust standard errors as well as Likelihood Ratio tes

# an example of a scan chromosome matrix
# desiged to eliminate duplicated individuals
# and scans with missing values of sex
library(GWASdata)
data(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
samp.chr.matrix <- matrix(TRUE,nrow(scanAnnot),26)
dup <- duplicated(scanAnnot$subjectID)
samp.chr.matrix[dup | is.na(scanAnnot$sex),] <- FALSE

# additionally, exclude YRI subjects
scan.exclude <- scanAnnot$scanID[scanAnnot$race == "YRI"]

# create some variables for the scans
scanAnnot$sex <- as.factor(scanAnnot$sex)
scanAnnot$age <- rnorm(nrow(scanAnnot),mean=40, sd=10)
scanAnnot$case.cntl.status <- rbinom(nrow(scanAnnot),1,0.4)
scanAnnot$blood.pressure[scanAnnot$case.cntl.status==1] <- rnorm(sum(scanAnnot$case.cntl.status==1),mean=100,sd
scanAnnot$blood.pressure[scanAnnot$case.cntl.status==0] <- rnorm(sum(scanAnnot$case.cntl.status==0),mean=90,sd=

# create data object
gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(gdsfile)
genoData <-  GenotypeData(gds, scanAnnot=scanAnnot)

# set regression variables and models
outcome <- c("case.cntl.status","blood.pressure")

covar.list <- list()
covar.list[[1]] <- c("age","sex")
covar.list[[2]] <- c("")
```

```
ivar.list <- list();
ivar.list[[1]] <- c("sex");
ivar.list[[2]] <- c("");

model.type <- c("logistic","linear")

gene.action.list <- list()
gene.action.list[[1]] <- c("additive")
gene.action.list[[2]] <- c("dominant", "recessive")

chr.set <- 21

outfile <- tempfile()

assocTestRegression(genoData,
                    outcome = outcome,
                    model.type = model.type,
                    covar.list = covar.list,
                    ivar.list = ivar.list,
                    gene.action.list = gene.action.list,
                    scan.chromosome.filter = samp.chr.matrix,
                    scan.exclude = scan.exclude,
                    CI = 0.95,
                    robust = TRUE,
                    LRtest = TRUE,
                    chromosome.set = chr.set,
                    outfile = outfile)

model1 <- getobj(paste(outfile, ".model.1.additive.chr.21_21.RData", sep=""))
model2 <- getobj(paste(outfile, ".model.2.dominant.chr.21_21.RData", sep=""))
model3 <- getobj(paste(outfile, ".model.2.recessive.chr.21_21.RData", sep=""))

close(genoData)
unlink(paste(outfile, "*", sep=""))

# In order to run the test on all chromosomes, it is suggested to run the function in parallel.
# To run the function in parallel the following unix can be used:
# R --vanilla --args 21 22 < assoc.analysis.r >logfile.txt &
# where the file assoc.analysis.r will include commands similar to this example
# where chromosome.set and/or block.set can be passed to R using --args
# Here, tests on chromosomes 21 and 22 are performed; these could be replaced by any set of chromosomes
# these values are retrieved in R by putting a
# chr.set <- as.numeric(commandArgs(trailingOnly=TRUE))
# command in assoc.analysis.r
```

BAFfromClusterMeans *B Allele Frequency & Log R Ratio Calculation*

Description

This function calculates the B allele frequency and the log R ratio values from the mean R and theta values for each cluster.

Usage

```
BAFfromClusterMeans(intenData, filename, file.type = c("gds", "ncdf"),
                    clusterMeanVars = c("tAA","tAB","tBB","rAA","rAB","rBB"),
            precision="single", compress="ZIP.max",
                    verbose = TRUE)
```

Arguments

intenData	IntensityData object holding the X and Y intensity data from which the B allele frequency and log R ratio are calculated.
filename	The name of the genotype GDS or netCDF file to create
file.type	The type of file to create ("gds" or "ncdf")
clusterMeanVars	Character vector indicating the names of the cluster mean columns in the SNP annotation of intenData. Must be in order (tAA,tAB,tBB,rAA,rAB,rBB).
precision	A character value indicating whether floating point numbers should be stored as "double" or "single" precision.
compress	The compression level for variables in a GDS file (see add.gdsn for options.
verbose	Logical value specifying whether to show progress information.

Details

This function calculates the B allele frequency and the log R ratio values from the mean R and theta values for each cluster and writes them to a GDS or NetCDF file.

Author(s)

Stephanie Gogarten, Caitlin McHugh

References

Peiffer D.A., Le J.M., Steemers F.J., Chang W., Jenniges T., and et al. High-resolution genomic profiling of chromosomal aberrations using infinium whole-genome genotyping. Genome Research, 16:1136-1148, 2006.

See Also

IntensityData, BAFfromClusterMeans

Examples

```
# create IntensityData object from GDS
library(GWASdata)
xyfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
xy <- GdsIntensityReader(xyfile)
data(illuminaSnpADF)
xyData <- IntensityData(xy, snpAnnot=illuminaSnpADF)

# calculate BAF and LRR and store in GDS file
blfile <- tempfile()
BAFfromClusterMeans(xyData, blfile, file.type="gds", verbose=FALSE)

# read output
bl <- GdsIntensityReader(blfile)
baf <- getBAlleleFreq(bl)
lrr <- getLogRRatio(bl)

close(xy)
close(bl)
file.remove(blfile)
```

BAFfromGenotypes *B Allele Frequency & Log R Ratio Calculation*

Description

This function calculates the B allele frequency and the log R ratio values for samples by either plate or by study.

Usage

```
BAFfromGenotypes(intenData, genoData,
                 filename, file.type = c("gds", "ncdf"),
                 min.n.genotypes = 2,
                 call.method = c("by.plate", "by.study"),
                 plate.name = "plate",
                 block.size = 5000,
         precision="single", compress="ZIP.max",
                 verbose = TRUE)
```

Arguments

intenData	IntensityData object holding the X and Y intensity data from which the B allele frequency and log R ratio are calculated.
genoData	GenotypeData object.
filename	The name of the genotype GDS or netCDF file to create
file.type	The type of file to create ("gds" or "ncdf")

`min.n.genotypes`
: The minimum number of samples for each genotype at any SNP in order to have non-missing B allele freqency and log R ratio. Setting this parameter to 2 or a similar value is recommended.

`call.method`
: If call.method is 'by.plate', the B allele frequency and log R ratio are calculated for samples delineated by plates. This is the default method. If call.method is 'by.study', the calculation uses all samples at once. If a study does not have plate specifications, 'by.study' is the call.method that must be used.

`plate.name`
: Character string specifying the name of the plate variable in intenData or genoData. By default, the plate.name is simply 'plate' but oftentimes there are variations, such as 'plateID' or 'plate.num'.

`block.size`
: An integer specifying the number of SNPs to be loaded at one time. The recommended value is around 1000, but should vary depending on computing power.

`precision`
: A character value indicating whether floating point numbers should be stored as "double" or "single" precision.

`compress`
: The compression level for variables in a GDS file (see `add.gdsn` for options.

`verbose`
: Logical value specifying whether to show progress information.

Details

Because this function can take a considerable amount of time and space, sufficient attention should be given to the value used for `block.size`.

Author(s)

Caitlin McHugh

References

Peiffer D.A., Le J.M., Steemers F.J., Chang W., Jenniges T., and et al. High-resolution genomic profiling of chromosomal aberrations using infinium whole-genome genotyping. Genome Research, 16:1136-1148, 2006.

See Also

`IntensityData`, `GenotypeData`, `chromIntensityPlot`, `BAFfromClusterMeans`

Examples

```
## Not run:
# create IntensityData and GenotypeData objects from netCDF
library(GWASdata)
data(affySnpADF)
data(affyScanADF)
nsamp <- nrow(affyScanADF)

xyfile <- system.file("extdata", "affy_qxy.nc", package="GWASdata")
xyNC <- NcdfIntensityReader(xyfile)
xyData <- IntensityData(xyNC, snpAnnot=affySnpADF, scanAnnot=affyScanADF)
```

```
genofile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
genoNC <- NcdfGenotypeReader(genofile)
genoData <- GenotypeData(genoNC, snpAnnot=affySnpADF, scanAnnot=affyScanADF)

# calculate BAF and LRR
blfile <- tempfile()
BAFfromGenotypes(xyData, genoData, blfile, file.type="ncdf", min.n.genotypes=2,
                 call.method="by.plate", plate.name="plate")

blNC <- NcdfIntensityReader(blfile)
baf <- getBAlleleFreq(blNC)
lrr <- getLogRRatio(blNC)

close(xyData)
close(genoData)
close(blNC)
file.remove(blfile)

## End(Not run)
```

---

batchTest *Batch Effects of Genotyping*

Description

batchChisqTest calculates Chi-square values for batches from 2-by-2 tables of SNPs, comparing each batch with the other batches. batchFisherTest calculates Fisher's exact test values.

Usage

```
batchChisqTest(genoData, batchVar,
               chrom.include = 1:22, sex.include = c("M", "F"),
               scan.exclude = NULL, return.by.snp = FALSE,
               correct = TRUE, verbose = TRUE,
               outfile = NULL)

batchFisherTest(genoData, batchVar,
                chrom.include = 1:22, sex.include = c("M", "F"),
                scan.exclude = NULL, return.by.snp = FALSE,
                conf.int = FALSE, verbose = TRUE,
                outfile = NULL)
```

Arguments

genoData	GenotypeData object
batchVar	A character string indicating which annotation variable should be used as the batch.

chrom.include	Integer vector with codes for chromosomes to include. Default is 1:22 (autosomes). Use 23, 24, 25, 26, 27 for X, XY, Y, M, Unmapped respectively
sex.include	Character vector with sex to include. Default is c("M", "F"). If sex chromosomes are present in chrom.include, only one sex is allowed.
scan.exclude	An integer vector containing the IDs of scans to be excluded.
return.by.snp	Logical value to indicate whether snp-by-batch matrices should be returned.
conf.int	Logical value to indicate if a confidence interval should be computed.
correct	Logical value to specify whether to apply the Yates continuity correction.
verbose	Logical value specifying whether to show progress information.
outfile	A character string to append in front of ".RData" for naming the output file.

Details

Because of potential batch effects due to sample processing and genotype calling, batches are an important experimental design factor.

batchChisqTest calculates the Chi square values from 2-by-2 table for each SNP, comparing each batch with the other batches.

batchFisherTest calculates Fisher's Exact Test from 2-by-2 table for each SNP, comparing each batch with the other batches.

For each SNP and each batch, batch effect is evaluated by a 2-by-2 table: # of A alleles, and # of B alleles in the batch, versus # of A alleles, and # of B alleles in the other batches. Monomorphic SNPs are set to NA for all batches.

The default behavior is to combine allele frequencies from males and females and return results for autosomes only. If results for sex chromosomes (X or Y) are desired, use chrom.include with values 23 and/or 25 and sex.include="M" or "F".

If there are only two batches, the calculation is only performed once and the values for each batch will be identical.

Value

If outfile=NULL (default), all results are returned as a list. If outfile is specified, no data is returned but the list is saved to disk as "outfile.RData."

batchChisqTest returns a list with the following elements:

mean.chisq	a vector of mean chi-squared values for each batch.
lambda	a vector of genomic inflation factor computed as median(chisq) / 0.456 for each batch.
chisq	a matrix of chi-squared values with SNPs as rows and batches as columns. Only returned if return.by.snp=TRUE.

batchFisherTest returns a list with the following elements:

mean.or	a vector of mean odds-ratio values for each batch. mean.or is computed as 1/mean(pmin(or, 1/or)) since the odds ratio is >1 when the batch has a higher allele frequency than the other batches and <1 for the reverse.

lambda	a vector of genomic inflation factor computed as median(-2*log(pval) / 1.39 for each batch.

Each of the following is a matrix with SNPs as rows and batches as columns, and is only returned if return.by.snp=TRUE:

pval	P value
oddsratio	Odds ratio
confint.low	Low value of the confidence interval for the odds ratio. Only returned if conf.int=TRUE.
confint.high	High value of the confidence interval for the odds ratio. Only returned if conf.int=TRUE.

batchChisqTest and batchFisherTest both also return the following if return.by.snp=TRUE:

allele.counts	matrix with total number of A and B alleles over all batches.
min.exp.freq	matrix of minimum expected allele frequency with SNPs as rows and batches as columns.

Warnings:

If outfile is not NULL, another file will be saved with the name "outfile.warnings.RData" that contains any warnings generated by the function.

Author(s)

Xiuwen Zheng, Stephanie Gogarten

See Also

GenotypeData, chisq.test, fisher.test

Examples

```
library(GWASdata)
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)
data(illuminaScanADF)
genoData <-  GenotypeData(gds, scanAnnot=illuminaScanADF)

# autosomes only, sexes combined (default)
res.chisq <- batchChisqTest(genoData, batchVar="plate")
res.chisq$mean.chisq
res.chisq$lambda

# X chromosome for females
res.chisq <- batchChisqTest(genoData, batchVar="status",
  chrom.include=23, sex.include="F", return.by.snp=TRUE)
head(res.chisq$chisq)

# Fisher exact test of "status" on X chromosome for females
res.fisher <- batchFisherTest(genoData, batchVar="status",
  chrom.include=23, sex.include="F", return.by.snp=TRUE)
qqPlot(res.fisher$pval)
```

```
close(genoData)
```

`centromeres` *Centromere base positions*

Description

Centromere base positions from the GRCh36/hg18 and GRCh37/hg19 genome builds.

Usage

```
data(centromeres.hg18)
data(centromeres.hg19)
```

Format

A data frame with the following columns.

`chrom` chromosome (1-22, X, Y)

`left.base` starting base position of centromere

`right.base` ending base position of centromere

Note

The UCSC genome browser lists two regions for the Y chromosome centromere in build hg18. We removed the positions (12208578, 12308578) from the centromere table to avoid problems with duplicate entries in the code.

Source

UCSC genome browser (http://genome.ucsc.edu).

Examples

```
data(centromeres.hg18)
data(centromeres.hg19)
```

chromIntensityPlot *Plot B Allele Frequency and/or Log R Ratio, R or Theta values for samples by probe position on a chromosome*

Description

This function creates plots for one or more of the 'B AlleleFreq', 'Log R Ratio', 'R' or 'Theta' values for given sample by chromosome combinations.

Usage

```
chromIntensityPlot(intenData, scan.ids, chrom.ids,
  type = c("BAF/LRR", "BAF", "LRR", "R", "Theta", "R/Theta"),
  main = NULL, info = NULL, abln = NULL,
  horizln = c(1/2, 1/3, 2/3),
  colorGenotypes = FALSE, genoData = NULL,
  colorBatch = FALSE, batch.column = NULL,
  snp.exclude = NULL,
  ideogram=TRUE, ideo.zoom=TRUE, ideo.rect=FALSE,
  cex=0.5, cex.leg=1.5, ...)
```

Arguments

intenData	IntensityData object, must contain at least one of 'BAlleleFreq', 'LogRRatio', 'X', 'Y'.
scan.ids	A vector containing the scan IDs to plot.
chrom.ids	A vector containing the chromosomes to plot for each scanID (should have same length as scan.ids).
type	The type of plot to be created. 'BAF/LRR' creates both 'B Allele Freq' and 'Log R Ratio' plots. 'R/Theta' creates both 'R' and 'Theta' plots.
main	Vector of plot titles. If NULL then the title will include scanID, sex, and chromosome.
info	A character vector containing extra information to include in the main title.
abln	A vector of values that is of length 2*length(scan.ids). Each pair of entries specifies where vertical lines will be drawn in each plot. This is especially useful when drawing the start \& end breakpoints for anomalies, for example. Use -1 as one pair value for plots that warrant only one line. By default, no lines will be drawn.
horizln	A vector containing the y-axis values at which a horizontal line will be drawn in B Allele Frequency plots.
colorGenotypes	A logical value specifying whether to color-code the points by called genotype. if TRUE, genoData must be given also.
genoData	GenotypeData object, required if colorGenotypes=TRUE.

colorBatch	A logical value specifying whether to color-code the points by sample batch (e.g, plate). If TRUE, batch.column must also be specified.
batch.column	A character string indicating which annotation variable in intenData should be used as the batch.
snp.exclude	An integer vector giving the IDs of SNPs to exclude from the plot.
ideogram	logical for whether to plot a chromosome ideogram under the BAF and LRR plots.
ideo.zoom	logical for whether to zoom in on the ideogram to match the range of the BAF/LRR plots.
ideo.rect	logical for whether to draw a rectangle on the ideogram indicating the range of the BAF/LRR plots.
cex	cex value for points on the plots.
cex.leg	cex value for the ideogram legend.
...	Other parameters to be passed directly to plot.

Details

For all plots, a vertical line is drawn every one eigth of the chromosome. For the Log R Ratio plot, the y-axis has been given the range of (-2,2).

Author(s)

Caitlin McHugh, Cathy Laurie

See Also

IntensityData, GenotypeData, BAFfromGenotypes

Examples

```
library(GWASdata)
data(illuminaScanADF)

blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
bl <- GdsIntensityReader(blfile)
intenData <-  IntensityData(bl, scanAnnot=illuminaScanADF)

genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <-  GenotypeData(geno, scanAnnot=illuminaScanADF)

scanID <- getScanID(illuminaScanADF, index=1)
chromIntensityPlot(intenData=intenData, scan.ids=scanID,
                   chrom.ids=22, type="BAF/LRR", info="interesting sample",
                   colorGenotypes=TRUE, genoData=genoData)
close(genoData)
close(intenData)
```

convertNcdfGds *Convert between NetCDF and GDS format*

Description

convertNcdfGds converts a NetCDF file to GDS format.

convertGdsNcdf converts a GDS file to NetCDF format.

checkNcdfGds checks whether a genotype NetCDF file and a GDS file contain identical data.

Usage

```
convertNcdfGds(ncdf.filename, gds.filename, snp.annot = NULL,
  precision = "single", compress = "ZIP.max", verbose = TRUE)

convertGdsNcdf(gds.filename, ncdf.filename,
  precision = "single", verbose = TRUE)

checkNcdfGds(ncdf.filename, gds.filename, verbose = TRUE)
```

Arguments

ncdf.filename	name of the NetCDF file
gds.filename	name of the GDS file
snp.annot	a SnpAnnotationDataFrame with SNP annotation. The column named "snpName" will be written to "snp.rs.id" in the GDS file.
precision	A character value indicating whether floating point numbers should be stored as "double" or "single" precision.
compress	the compression format for the GDS file, one of "", "ZIP", "ZIP.fast", "ZIP.default", or "ZIP.max"
verbose	whether to show progress information

Details

convertNcdfGds assumes any variables other than "sampleID", "chromosome", and "position" have dimensions SNP x sample.

If snp.annot has columns "rsID", "alleleA", "alleleB", these will be stored in the GDS file as "snp.rs.id" and "snp.allele" (the latter has the format "A/B").

Chromosome codes from snp.annot (for autosomes, X, Y, etc.) will be stored in the GDS file.

convertGdsNcdf assumes any variables not starting with "snp" or "sample" have dimensions SNP x sample.

Value

checkNcdfGds returns TRUE if the NetCDF and GDS files contain identical data. If the files differ, it will print a diagnostic message and return FALSE.

Author(s)

Xiuwen Zheng, Stephanie Gogarten

See Also

gdsfmt, **ncdf**

Examples

```
library(GWASdata)
ncfile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")

data(illuminaSnpADF)

gdsfile <- tempfile()
convertNcdfGds(ncfile, gdsfile, snp.annot=illuminaSnpADF)

checkNcdfGds(ncfile, gdsfile)

ncfile2 <- tempfile()
convertGdsNcdf(gdsfile, ncfile2)

file.remove(gdsfile, ncfile2)
```

---

convertVcfGds *Conversion from VCF to GDS*

Description

Extract SNP data from a VCF file

Usage

```
convertVcfGds(vcf.filename, gds.filename, nblock=1024, compress="ZIP.max",
      verbose=TRUE)
```

Arguments

vcf.filename	the file name of VCF format
gds.filename	the output gds file
nblock	the buffer lines
compress	the compression format for the GDS file, one of "", "ZIP", "ZIP.fast", "ZIP.default", or "ZIP.max"
verbose	whether to show progress information

Details

convertVcfGds extracts bi-allelic SNP genotypes from a VCF file and stores them in a GDS file. All VCF rows which do not contain polymorphic, bi-allelic SNPs are ignored. Unique integer IDs are generated for all samples and SNPs. Sample name, SNP ID, reference and alternate alleles, chromosome, and position are stored in the GDS file as well.

GDS – Genomic Data Structures, the extended file name used for storing genetic data, and the file format used in the **gdsfmt** package.

VCF – The Variant Call Format (VCF), which is a generic format for storing DNA polymorphism data such as SNPs, insertions, deletions and structural variants, together with rich annotations.

Author(s)

Xiuwen Zheng

References

The variant call format and VCFtools. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. Bioinformatics. 2011 Aug 1;27(15):2156-8. Epub 2011 Jun 7.

http://corearray.sourceforge.net/

See Also

GdsGenotypeReader

Examples

```
# The VCF file
vcf.file <- system.file("extdata", "sequence.vcf", package="SNPRelate")
readLines(vcf.file)

gds.file <- tempfile()
convertVcfGds(vcf.file, gds.file)

# open GDS file
(gds <- GdsGenotypeReader(gds.file))

getScanID(gds)
getSnpID(gds)
getChromosome(gds)
getPosition(gds)
getVariable(gds, "sample.name")
getVariable(gds, "snp.rs.id")
getVariable(gds, "snp.allele")
getGenotype(gds)

# close the genotype file
close(gds)
unlink(gds.file)
```

createDataFile *Write genotypic calls and/or associated metrics to a GDS or netCDF file.*

Description

Genotypic calls and/or associated quantitative variables (e.g. quality score, intensities) are read from text files and written to a GDS or netCDF file.

Usage

```
createDataFile(path = ".", filename, file.type=c("gds", "ncdf"),
               variables="genotype", snp.annotation, scan.annotation,
               sep.type, skip.num, col.total, col.nums, scan.name.in.file,
       precision="single", compress="ZIP.max",
       array.name = NULL, genome.build = NULL,
               diagnostics.filename = "createDataFile.diagnostics.RData",
               verbose = TRUE)

createAffyIntensityFile(path = ".",  filename, file.type=c("gds", "ncdf"),
                        snp.annotation, scan.annotation,
                precision="single", compress="ZIP.max",
                array.name = NULL, genome.build = NULL,
             diagnostics.filename = "createAffyIntensityFile.diagnostics.RData",
                  verbose = TRUE)

checkGenotypeFile(path = ".", filename, file.type=c("gds", "ncdf"),
                  snp.annotation, scan.annotation,
                  sep.type, skip.num, col.total, col.nums, scan.name.in.file,
  check.scan.index, n.scans.loaded,
                  diagnostics.filename = "checkGenotypeFile.diagnostics.RData",
                  verbose = TRUE)

checkIntensityFile(path = ".", filename, file.type=c("gds", "ncdf"),
                   snp.annotation, scan.annotation,
                   sep.type, skip.num, col.total, col.nums, scan.name.in.file,
   check.scan.index, n.scans.loaded, affy.inten = FALSE,
                 diagnostics.filename = "checkIntensityFile.diagnostics.RData",
                   verbose = TRUE)
```

Arguments

path	Path to the raw text files.
filename	The name of the genotype GDS or netCDF file to create
file.type	The type of file to create ("gds" or "ncdf")

`variables` A character vector containing the names of the variables to create (must be one or more of `c("genotype", "quality", "X", "Y", "rawX", "rawY", "R", "Theta", "BAlleleFreq"`

`snp.annotation` Snp annotation dataframe with columns "snpID", "chromosome", "position" and "snpName". snpID should be a unique integer vector, sorted with respect to chromosome and position. snpName should match the snp identifiers inside the raw genoypic data files If `file.type="gds"`, optional columns "alleleA", and "alleleB" will be written if present.

`scan.annotation`
Scan annotation data.frame with columns "scanID" (integer id of genotyping instance), "scanName", (sample name inside the raw data file) and "file" (corresponding raw data file name).

`sep.type` Field separator in the raw text files.

`skip.num` Number of rows to skip, which should be all rows preceding the genotypic or quantitative data (including the header).

`col.total` Total number of columns in the raw text files.

`col.nums` An integer vector indicating which columns of the raw text file contain variables for input. `names(col.nums)` must be a subset of c("snp", "sample", "geno", "a1", "a2", "quality", "X", "Y", "rawX", "rawY", "R", "Theta", "BAlleleFreq", "LogRRatio"). The element "snp" is the column of SNP ids, "sample" is sample ids, "geno" is diploid genotype (in AB format), "a1" and "a2" are alleles 1 and 2 (in AB format), "quality" is quality score, "X" and "Y" are normalized intensities, "rawX" and "rawY" are raw intensities, "R" is the sum of normalized intensities, "Theta" is angular polar coordinate, "BAlleleFreq" is the B allele frequency, and "LogRRatio" is the Log R Ratio.

`scan.name.in.file`
An indicator for the presence of sample name within the file. A value of 1 indicates a column with repeated values of the sample name (Illumina format), -1 indicates sample name embedded in a column heading (Affymetrix format) and 0 indicates no sample name inside the raw data file.

`check.scan.index`
An integer vector containing the indices of the sample dimension of the GDS or netCDF file to check.

`n.scans.loaded` Number of scans loaded in the GDS or netCDF file.

`affy.inten` Logical value indicating whether intensity files are in Affymetrix format (two lines per SNP).

`precision` A character value indicating whether floating point numbers should be stored as "double" or "single" precision.

`compress` The compression level for variables in a GDS file (see `add.gdsn` for options.

`array.name` Name of the array, to be stored as an attribute in the netCDF file.

`genome.build` Genome build used in determining chromosome and position, to be stored as an attribute in the netCDF file.

`diagnostics.filename`
Name of the output file to save diagnostics.

`verbose` Logical value specifying whether to show progress information.

Details

These functions read genotypic and associated data from raw text files. The files to be read and processed are specified in the sample annotation. `createDataFile` expects one file per sample, with each file having one row of data per SNP probe. The `col.nums` argument allows the user to select and identify specific fields for writing to the GDS or netCDF file. Illumina text files and Affymetrix ".CHP" files can be used here (but not Affymetrix "ALLELE_SUMMARY" files).

A SNP annotation data.frame is a pre-requisite for this function. It has the same number of rows (one per SNP) as the raw text file and a column of SNP names matching those within the raw text file. It also has a column of integer SNP ids to be used as a unique key for each SNP in the GDS or netCDF file.

A sample annotation data.frame is also a pre-requisite. It has one row per sample with columns corresponding to sample name (as it occurs within the raw text file), name of the raw text file for that sample and an integer sample id (to be written as the "sampleID" variable in the GDS or netCDF file).

The genotype calls in the raw text file may be either one column of diploid calls or two columns of allele calls. The function takes calls in AB format and converts them to a numeric code indicating the number of "A" alleles in the genotype (i.e. AA=2, AB=1, BB=0 and missing=-1).

While each raw text file is being read, the functions check for errors and irregularities and records the results in a list of vectors. If any problem is detected, that raw text file is skipped.

`createAffyIntensityFile` create an intensity data file from Affymetrix "ALLELE_SUMMARY" files. The "ALLELE_SUMMARY" files have two rows per SNP, one for X (A allele) and one for Y (B allele). These are reformatted to one row per SNP and and ordered according to the SNP integer id. The correspondence between SNP names in the "ALLELE_SUMMARY" file and the SNP integer ids is made using the SNP annotation data.frame.

`checkGenotypeFile` and `checkIntensityFile` check the contents of GDS or netCDF files against raw text files.

Value

The GDS or netCDF file specified in argument `filename` is populated with genotype calls and/or associated quantitative variables. A list of diagnostics with the following components is returned. Each vector has one element per raw text file processed.

`read.file`	A vector indicating whether (1) or not (0) each file was read successfully.
`row.num`	A vector of the number of rows read from each file. These should all be the same and equal to the number of rows in the SNP annotation data.frame.
`samples`	A list of vectors containing the unique sample names in the sample column of each raw text file. Each vector should have just one element.
`sample.match`	A vector indicating whether (1) or not (0) the sample name inside the raw text file matches that in the sample annotation data.frame
`missg`	A list of vectors containing the unique character string(s) for missing genotypes (i.e. not AA,AB or BB) for each raw text file.
`snp.chk`	A vector indicating whether (1) or not (0) the raw text file has the expected set of SNP names (i.e. matching those in the SNP annotation data.frame).

chk — A vector indicating whether (1) or not (0) all previous checks were successful and the data were written to the netCDF file.

checkGenotypeFile returns the following additional list items.

snp.order — A vector indicating whether (1) or not (0) the snp ids are in the same order in each file.

geno.chk — A vector indicating whether (1) or not (0) the genotypes in the netCDF match the text file.

checkIntensityFile returns the following additional list items.

qs.chk — A vector indicating whether (1) or not (0) the quality scores in the netCDF match the text file.

read.file.inten — A vector indicating whether (1) or not (0) each intensity file was read successfully (if intensity files are separate).

sample.match.inten — A vector indicating whether (1) or not (0) the sample name inside the raw text file matches that in the sample annotation data.frame (if intensity files are separate).

rows.equal — A vector indicating whether (1) or not (0) the number of rows read from each file are the same and equal to the number of rows in the SNP annotation data.frame (if intensity files are separate).

snp.chk.inten — A vector indicating whether (1) or not (0) the raw text file has the expected set of SNP names (i.e. matching those in the SNP annotation data.frame) (if intensity files are separate).

inten.chk — A vector for each intensity variable indicating whether (1) or not (0) the intensities in the netCDF match the text file.

Author(s)

Stephanie Gogarten, Cathy Laurie

See Also

gdsfmt, ncdf

Examples

```
library(GWASdata)

#############
# Illumina - genotype file
#############
gdsfile <- tempfile()
path <- system.file("extdata", "illumina_raw_data", package="GWASdata")
data(illumina_snp_annot, illumina_scan_annot)
snpAnnot <- illumina_snp_annot[,c("snpID", "rsID", "chromosome",
                          "position", "alleleA", "alleleB")]
```

```
names(snpAnnot)[2] <-  "snpName"
# subset of samples for testing
scanAnnot <- illumina_scan_annot[1:3, c("scanID", "genoRunID", "file")]
names(scanAnnot)[2] <- "scanName"
col.nums <- as.integer(c(1,2,12,13))
names(col.nums) <- c("snp", "sample", "a1", "a2")
diagfile <- tempfile()
res <- createDataFile(path, gdsfile, file.type="gds", variables="genotype",
                      snpAnnot, scanAnnot, sep.type=",",
                      skip.num=11, col.total=21, col.nums=col.nums,
                      scan.name.in.file=1, diagnostics.filename=diagfile)

file.remove(diagfile)
file.remove(gdsfile)


#############
# Affymetrix - genotype file
#############
gdsfile <- tempfile()
path <- system.file("extdata", "affy_raw_data", package="GWASdata")
data(affy_snp_annot, affy_scan_annot)
snpAnnot <- affy_snp_annot[,c("snpID", "probeID", "chromosome", "position")]
names(snpAnnot)[2] <- "snpName"
# subset of samples for testing
scanAnnot <- affy_scan_annot[1:3, c("scanID", "genoRunID", "chpFile")]
names(scanAnnot)[2:3] <- c("scanName", "file")
col.nums <- as.integer(c(2,3)); names(col.nums) <- c("snp", "geno")
diagfile <- tempfile()
res <- createDataFile(path, gdsfile, file.type="gds", variables="genotype",
                      snpAnnot, scanAnnot, sep.type="\t",
                      skip.num=1, col.total=6, col.nums=col.nums,
                      scan.name.in.file=-1, diagnostics.filename=diagfile)
file.remove(diagfile)

# check
diagfile <- tempfile()
res <- checkGenotypeFile(path, gdsfile, file.type="gds", snpAnnot, scanAnnot,
                        sep.type="\t", skip.num=1, col.total=6, col.nums=col.nums,
scan.name.in.file=-1,
check.scan.index=1:3, n.scans.loaded=3,
diagnostics.filename=diagfile)
file.remove(diagfile)
file.remove(gdsfile)


#############
# Affymetrix - intensity file
#############
gdsfile <- tempfile()
path <- system.file("extdata", "affy_raw_data", package="GWASdata")
data(affy_snp_annot, affy_scan_annot)
snpAnnot <- affy_snp_annot[,c("snpID", "probeID", "chromosome", "position")]
```

```
names(snpAnnot)[2] <- "snpName"
# subset of samples for testing
scanAnnot <- affy_scan_annot[1:3, c("scanID", "genoRunID", "alleleFile")]
names(scanAnnot)[2:3] <- c("scanName", "file")
diagfile <- tempfile()
res <- createAffyIntensityFile(path, gdsfile, file.type="gds", snpAnnot, scanAnnot,
            diagnostics.filename=diagfile)
file.remove(diagfile)

# check
diagfile <- tempfile()
res <- checkIntensityFile(path, gdsfile, file.type="gds", snpAnnot, scanAnnot,
                          sep.type="\t", skip.num=1, col.total=2,
  col.nums=setNames(as.integer(c(1,2,2)), c("snp", "X", "Y")),
     scan.name.in.file=-1, affy.inten=TRUE,
                          check.scan.index=1:3, n.scans.loaded=3,
      diagnostics.filename=diagfile)
file.remove(diagfile)
file.remove(gdsfile)
```

`duplicateDiscordance` *Duplicate discordance*

Description

A function to compute pair-wise genotype discordances between multiple genotyping instances of the same subject.

Usage

```
duplicateDiscordance(genoData, subjName.col,
                     one.pair.per.subj=TRUE, corr.by.snp=FALSE,
                     minor.allele.only=FALSE, allele.freq=NULL,
                     scan.exclude=NULL, snp.exclude=NULL,
                     verbose=TRUE)
```

Arguments

`genoData`	`GenotypeData` object
`subjName.col`	A character string indicating the name of the annotation variable that will be identical for duplicate scans.
`one.pair.per.subj`	A logical indicating whether a single pair of scans should be randomly selected for each subject with more than 2 scans.
`corr.by.snp`	A logical indicating whether correlation by SNP should be computed (may significantly increase run time).

`minor.allele.only`
: A logical indicating whether discordance should be calculated only between pairs of scans in which at least one scan has a genotype with the minor allele (i.e., exclude major allele homozygotes).

`allele.freq`
: A numeric vector with the frequency of the A allele for each SNP in `genoData`. Required if `minor.allele.only=TRUE`.

`scan.exclude`
: An integer vector containing the ids of scans to be excluded.

`snp.exclude`
: An integer vector containing the ids of SNPs to be excluded.

`verbose`
: Logical value specifying whether to show progress information.

Details

`duplicateDiscordance` calculates discordance metrics both by scan and by SNP. If `one.pair.per.subj=TRUE` (the default), each subject with more than two duplicate genotyping instances will have two scans randomly selected for computing discordance. If `one.pair.per.subj=FALSE`, discordances will be calculated pair-wise for all possible pairs for each subject.

Value

A list with the following components:

`discordance.by.snp`
: data frame with 5 columns: 1. snpID, 2. discordant (number of discordant pairs), 3. npair (number of pairs examined), 4. n.disc.subj (number of subjects with at least one discordance), 5. discord.rate (discordance rate i.e. discordant/npair)

`discordance.by.subject`
: a list of matrices (one for each subject) with the pair-wise discordance between the different genotyping instances of the subject

`correlation.by.subject`
: a list of matrices (one for each subject) with the pair-wise correlation between the different genotyping instances of thesubject

If `corr.by.snp=TRUE`, `discordance.by.snp` will also have a column "correlation" with the correlation between duplicate subjects. For this calculation, the first two samples per subject are selected.

Author(s)

Tushar Bhangale, Cathy Laurie, Stephanie Gogarten

See Also

`GenotypeData`, `duplicateDiscordanceAcrossDatasets`, `duplicateDiscordanceProbability`, `alleleFrequency`

Examples

```
library(GWASdata)
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)
data(illuminaScanADF)
genoData <-  GenotypeData(gds, scanAnnot=illuminaScanADF)

disc <- duplicateDiscordance(genoData, subjName.col="subjectID")

# minor allele discordance
afreq <- alleleFrequency(genoData)
minor.disc <- duplicateDiscordance(genoData, subjName.col="subjectID",
  minor.allele.only=TRUE, allele.freq=afreq[,"all"])

close(genoData)
```

---

duplicateDiscordanceAcrossDatasets

Duplicate discordance across datasets

Description

Finds number of discordant genotypes by SNP in pairs of duplicate scans of the same subject across multiple datasets.

Usage

```
duplicateDiscordanceAcrossDatasets(genoData1, genoData2,
  match.snps.on=c("position", "alleles"),
  subjName.cols, snpName.cols=NULL,
  one.pair.per.subj=TRUE, minor.allele.only=FALSE,
  missing.fail=c(FALSE, FALSE),
  scan.exclude1=NULL, scan.exclude2=NULL,
  snp.exclude1=NULL, snp.exclude2=NULL,
  snp.include=NULL,
  verbose=TRUE)

minorAlleleDetectionAccuracy(genoData1, genoData2,
  match.snps.on=c("position", "alleles"),
  subjName.cols, snpName.cols=NULL,
  missing.fail=TRUE,
  scan.exclude1=NULL, scan.exclude2=NULL,
  snp.exclude1=NULL, snp.exclude2=NULL,
  snp.include=NULL,
  verbose=TRUE)
```

Arguments

genoData1	GenotypeData object containing the first dataset.
genoData2	GenotypeData object containing the second dataset.
match.snps.on	One or more of ("position", "alleles", "name") indicating how to match SNPs. "position" will match SNPs on chromosome and position, "alleles" will also require the same alleles (but A/B designations need not be the same), and "name" will match on the columns give in snpName.cols.
subjName.cols	2-element character vector indicating the names of the annotation variables that will be identical for duplicate scans in the two datasets. (Alternatively, one character value that will be recycled).
snpName.cols	2-element character vector indicating the names of the annotation variables that will be identical for the same SNPs in the two datasets. (Alternatively, one character value that will be recycled).
one.pair.per.subj	A logical indicating whether a single pair of scans should be randomly selected for each subject with more than 2 scans.
minor.allele.only	A logical indicating whether discordance should be calculated only between pairs of scans in which at least one scan has a genotype with the minor allele (i.e., exclude major allele homozygotes).
missing.fail	For duplicateDiscordanceAcrossDatasets, a 2-element logical vector indicating whether missing values in datasets 1 and 2, respectively, will be considered failures (discordances with called genotypes in the other dataset). For minorAlleleDetectionAccuracy, a single logical indicating whether missing values in dataset 2 will be considered false negatives (missing.fail=TRUE) or ignored (missing.fail=FALSE).
scan.exclude1	An integer vector containing the ids of scans to be excluded from the first dataset.
scan.exclude2	An integer vector containing the ids of scans to be excluded from the second dataset.
snp.exclude1	An integer vector containing the ids of snps to be excluded from the first dataset.
snp.exclude2	An integer vector containing the ids of snps to be excluded from the second dataset.
snp.include	List of SNPs to include in the comparison. Should match the contents of the columns referred to by snpName.cols. Only valid if match.snps.on includes "name".
verbose	Logical value specifying whether to show progress information.

Details

duplicateDiscordanceAcrossDatasets calculates discordance metrics both by scan and by SNP. If one.pair.per.subj=TRUE (the default), each subject with more than two duplicate genotyping instances will have one scan from each dataset randomly selected for computing discordance. If one.pair.per.subj=FALSE, discordances will be calculated pair-wise for all possible cross-dataset pairs for each subject.

If `minor.allele.only=TRUE`, the allele frequency will be calculated in `genoData1`, using only samples common to both datasets.

If `snp.include=NULL` (the default), discordances will be found for all SNPs common to both datasets.

`genoData1` and `genoData2` should each have "alleleA" and "alleleB" defined in their SNP annotation. If allele coding cannot be found, the two datasets are assumed to have identical coding.

`minorAlleleDetectionAccuracy` summarizes the accuracy of minor allele detection in `genoData2` with respect to `genoData1` (the "gold standard"). `TP`=number of true positives, `TN`=number of true negatives, `FP`=number of false positives, and `FN`=number of false negatives. Accuracy is represented by four metrics:

- sensitivity for each SNP as `TP/(TP+FN)`
- specificity for each SNP as `TN/(TN+FP)`
- positive predictive value for each SNP as `TP/(TP+FP)`
- negative predictive value for each SNP as `TN/(TN+FN)`.

`TP`, `TN`, `FP`, and `FN` are calculated as follows:

		genoData1		
		mm	Mm	MM
	mm	2TP	1TP + 1FP	2FP
genoData2	Mm	1TP + 1FN	1TN + 1TP	1TN + 1FP
	MM	2FN	1FN + 1TN	2TN
	–	2FN	1FN	

"M" is the major allele and "m" is the minor allele (as calculated in `genoData1`). "-" is a missing call in `genoData2`. Missing calls in `genoData1` are ignored. If `missing.fail=FALSE`, missing calls in `genoData2` (the last row of the table) are also ignored.

Value

`duplicateDiscordanceAcrossDatasets` returns a list with the following components:

`discordance.by.snp`
: data frame with 4 columns: 1. discordant (number of discordant pairs), 2. npair (number of pairs examined), 3. n.disc.subj (number of subjects with at least one discordance), 4. discord.rate (discordance rate i.e. discordant/npair). Row names are the common snp ID.

`discordance.by.subject`
: a list of matrices (one for each subject) with the pair-wise discordance between the different genotyping instances of the subject

`minorAlleleDetectionAccuracy` returns a data.frame with the following columns. Row names are the common snp ID.

`npair`	number of sample pairs compared (non-missing in `genoData1`)
`sensitivity`	sensitivity

`specificity` specificity

`positivePredictiveValue`
Positive predictive value

`negativePredictiveValue`
Negative predictive value

If no duplicate scans or no common SNPs are found, these functions issue a warning message and return `NULL`.

Author(s)

Stephanie Gogarten, Jess Shen

See Also

GenotypeData, duplicateDiscordance, duplicateDiscordanceProbability

Examples

```
# first set
snp1 <- data.frame(snpID=1:10, chromosome=1L, position=101:110,
                   rsID=paste("rs", 101:110, sep=""),
                   alleleA="A", alleleB="G", stringsAsFactors=FALSE)
scan1 <- data.frame(scanID=1:3, subjectID=c("A","B","C"), sex="F", stringsAsFactors=FALSE)
mgr <- MatrixGenotypeReader(genotype=matrix(c(0,1,2), ncol=3, nrow=10), snpID=snp1$snpID,
                            chromosome=snp1$chromosome, position=snp1$position, scanID=1:3)
genoData1 <- GenotypeData(mgr, snpAnnot=SnpAnnotationDataFrame(snp1),
                          scanAnnot=ScanAnnotationDataFrame(scan1))

# second set
snp2 <- data.frame(snpID=1:5, chromosome=1L,
                   position=as.integer(c(101,103,105,107,107)),
                   rsID=c("rs101", "rs103", "rs105", "rs107", "rsXXX"),
                   alleleA= c("A","C","G","A","A"),
                   alleleB=c("G","T","A","G","G"),
                   stringsAsFactors=FALSE)
scan2 <- data.frame(scanID=1:3, subjectID=c("A","C","C"), sex="F", stringsAsFactors=FALSE)
mgr <- MatrixGenotypeReader(genotype=matrix(c(1,2,0), ncol=3, nrow=5), snpID=snp2$snpID,
                            chromosome=snp2$chromosome, position=snp2$position, scanID=1:3)
genoData2 <- GenotypeData(mgr, snpAnnot=SnpAnnotationDataFrame(snp2),
                          scanAnnot=ScanAnnotationDataFrame(scan2))

duplicateDiscordanceAcrossDatasets(genoData1, genoData2,
  match.snps.on="position",
  subjName.cols="subjectID")

duplicateDiscordanceAcrossDatasets(genoData1, genoData2,
  match.snps.on=c("position", "alleles"),
  subjName.cols="subjectID")

duplicateDiscordanceAcrossDatasets(genoData1, genoData2,
  match.snps.on=c("position", "alleles", "name"),
```

```
  subjName.cols="subjectID",
  snpName.cols="rsID")

duplicateDiscordanceAcrossDatasets(genoData1, genoData2,
  subjName.cols="subjectID",
  one.pair.per.subj=FALSE)

minorAlleleDetectionAccuracy(genoData1, genoData2,
  subjName.cols="subjectID")
```

`duplicateDiscordanceProbability` *Probability of duplicate discordance*

Description

`duplicateDiscordanceProbability` calculates the probability of observing discordant genotypes for duplicate samples.

Usage

```
duplicateDiscordanceProbability(npair,
                                error.rate = c(1e-5, 1e-4, 1e-3, 1e-2),
                                max.disc = 7)
```

Arguments

`npair`	The number of pairs of duplicate samples.
`error.rate`	A numeric vector of error rates (i.e., the rate at which a genotype will be called incorrectly).
`max.disc`	The maximum number of discordances for which to compute the probability.

Details

Since there are three possible genotypes, one call is correct and the other two are erroneous, so theoretically there are two error rates, a and b. The probability that duplicate genotyping instances of the same subject will give a discordant genotype is 2[(1 - a - b)(a + b) + ab]. When a and b are very small, this is approximately 2(a + b) or twice the total error rate. This function assumes that a == b, and the argument `error.rate` is the total error rate a + b.

Any negative values for the probability (due to precision problems for very small numbers) are set to 0.

Value

This function returns a matrix of probabilities, where the column names are error rates and the row names are expected number of discordant genotypes (>0 through >`max.disc`).

Author(s)

Cathy Laurie

See Also

duplicateDiscordance, duplicateDiscordanceAcrossDatasets

Examples

```
disc <- duplicateDiscordanceProbability(npair=10, error.rate=c(1e-6, 1e-4))

#probability of observing >0 discordant genotypes given an error rate 1e-6
disc[1,1]

#probability of observing >1 discordant genotypes given an error rate 1e-4
disc[2,2]
```

---

findBAFvariance — *Find chromosomal areas with high BAlleleFreq (or LogRRatio) standard deviation*

Description

sdByScanChromWindow uses a sliding window algorithm to calculate the standard deviation of the BAlleleFreq (or LogRRatio) values for a user specified number of bins across each chromosome of each scan.

medianSdOverAutosomes calculates the median of the BAlleleFreq (or LogRRatio) standard deviation over all autosomes for each scan.

meanSdByChromWindow calculates the mean and standard deviation of the BAlleleFreq standard deviation in each window in each chromosome over all scans.

findBAFvariance flags chromosomal areas with high BAlleleFreq standard deviation using previously calculated means and standard deviations over scans, typically results from sdByScanChromWindow.

Usage

```
sdByScanChromWindow(intenData, genoData=NULL, var="BAlleleFreq", nbins=NULL,
  snp.exclude=NULL, return.mean=FALSE, incl.miss=TRUE, incl.het=TRUE, incl.hom=FALSE)

medianSdOverAutosomes(sd.by.scan.chrom.window)

meanSdByChromWindow(sd.by.scan.chrom.window, sex)

findBAFvariance(sd.by.chrom.window, sd.by.scan.chrom.window,
  sex, sd.threshold)
```

Arguments

intenData	A IntensityData object. The order of SNPs is expected to be by chromosome and then by position within chromosome.
genoData	A GenotypeData object. May be omitted if incl.miss, incl.het, and incl.hom are all TRUE, as there is no need to distinguish between genotype calls in that case.
var	The variable for which to calculate standard deviations, typically "BAlleleFreq" (the default) or "LogRRatio."
nbins	A vector with integers corresponding to the number of bins for each chromosome. The values all must be even integers.
snp.exclude	An integer vector containing the snpIDs of SNPs to be excluded.
return.mean	a logical. If TRUE, return mean as well as standard deviation.
incl.miss	a logical. If TRUE, include SNPs with missing genotype calls.
incl.het	a logical. If TRUE, include SNPs called as heterozygotes.
incl.hom	a logical. If TRUE, include SNPs called as homozygotes. This is typically FALSE (the default) for BAlleleFreq calculations.
sd.by.scan.chrom.window	A list of matrices of standard deviation for each chromosome, with dimensions of number of scans x number of windows. This is typically the output of sdByScanChromWindow.
sd.by.chrom.window	A list of matrices of the standard deviations, as generated by meanSdByChromWindow.
sex	A character vector of sex ("M"/"F") for the scans.
sd.threshold	A value specifying the threshold for the number of standard deviations above the mean at which to flag.

Details

sdByScanChromWindow calculates the standard deviation of BAlleleFreq (or LogRRatio) values across chromosomes 1-22 and chromosome X for a specified number of 'bins' in each chromosome as passed to the function in the 'nbins' argument. The standard deviation is calculated using windows of width equal to 2 bins, and moves along the chromosome by an offset of 1 bin (or half a window). Thus, there will be a total of nbins-1 windows per chromosome. If nbins=NULL (the default), there will be 2 bins (one window) for each chromosome.

medianSdOverAutosomes calulates the median over autosomes of BAlleleFreq (or LogRRatio) standard deviations calculated for sliding windows within each chromosome of each scan. The standard deviations should be a list with one element for each chromosome, and each element consisting of a matrix with scans as rows.

meanSdByChromWindow calculates the mean and standard deviation over scans of BAlleleFreq standard deviations calculated for sliding windows within each chromosome of each scan. The BAlleleFreq standard deviations should be a list with one element for each chromosome, and each element consisting of a matrix containing the BAlleleFreq standard deviation for the i'th scan in the j'th bin. This is typically created using the sdByScanChromWindow function. For the X chromosome the calculations are separated out by sex.

findBAFvariance determines which chromosomes of which scans have regions which are at least a given number of SDs from the mean, using BAlleleFreq means and standard deviations calculated from sliding windows over each chromosome by scan.

Value

sdByScanChromWindow returns a list of matrices containing standard deviations. There is a matrix for each chromosome, with each matrix having dimensions of number of scans x number of windows. If return.mean=TRUE, two lists to matrices are returned, one with standard deviations and one with means.

medianSdOverAutosomes returns a data frame with colums "scanID" and "med.sd" containing the median standard deviations over all autosomes for each scan.

meanSdByChromWindow returns a list of matrices, one for each chromosome. Each matrix contains two columns called "Mean" and "SD", containing the mean and SD of the BAlleleFreq standard devations over scans for each bin. For the X chromosome the matrix has four columns "Female Mean", "Male Mean", "Female SD" and "Male SD".

findBAFvariance returns a matrix with columns "scanID", "chromosome", "bin", and "sex" containing those scan by chromosome combinations with BAlleleFreq standard deviations greater than those specified by sd.threshold.

Author(s)

Caitlin McHugh, Cathy Laurie

See Also

IntensityData, GenotypeData, BAFfromClusterMeans, BAFfromGenotypes

Examples

```
library(GWASdata)
data(illuminaScanADF)

blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
bl <- GdsIntensityReader(blfile)
blData <-  IntensityData(bl, scanAnnot=illuminaScanADF)

genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <-  GenotypeData(geno, scanAnnot=illuminaScanADF)

nbins <- rep(8, 3) # need bins for chromosomes 21,22,23
baf.sd <- sdByScanChromWindow(blData, genoData, nbins=nbins)

close(blData)
close(genoData)
med.res <- medianSdOverAutosomes(baf.sd)

sex <- illuminaScanADF$sex
sd.res <- meanSdByChromWindow(baf.sd, sex)
```

```
var.res <- findBAFvariance(sd.res, baf.sd, sex, sd.threshold=2)
```

GdsGenotypeReader *Class GdsGenotypeReader*

Description

The GdsGenotypeReader class is an extension of the GdsReader class specific to reading genotype data stored in GDS files. GDS files with both `snp x scan` and `scan x snp` dimensions are supported.

Extends

GdsReader

Constructor

`GdsGenotypeReader(filename, genotypeDim)`:

`filename` must be the path to a GDS file or a gds object. The GDS file must contain the following variables:

- 'snp.id': a unique integer vector of snp ids
- 'snp.chromosome': integer chromosome codes
- 'snp.position': integer position values
- 'sample.id': a unique integer vector of scan ids
- 'genotype': a matrix of bytes with dimensions ('snp','sample'). The byte values must be the number of A alleles : 2=AA, 1=AB, 0=BB.

The optional variable "snp.allele" stores the A and B alleles in a character vector with format "A/B".

Default values for chromosome codes are 1-22=autosome, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `autosomeCode`, `XchromCode`, `XYchromCode`, `YchromCode`, and `MchromCode`.

The constructor automatically detects whether the GDS file is in `snp x scan` or `scan x snp` order using the dimensions of `snp.id` and `sample.id`. In the case of GDS files with equal SNP and scan dimensions, `genotypeDim` is a required input to the function and can take values `"snp,scan"` or `"scan,snp"`.

The `GdsGenotypeReader` constructor creates and returns a GdsGenotypeReader instance pointing to this file.

Accessors

In the code snippets below, `object` is a GdsGenotypeReader object. `snp` and `scan` indicate which elements to return along the snp and scan dimensions. They must be integer vectors of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that the entire dimension should be read. If snp and/or is scan omitted, the entire variable is read.

See GdsReader for additional methods.

`nsnp(object)`: The number of SNPs in the GDS file.

`nscan(object)`: The number of scans in the GDS file.

`getSnpID(object, index)`: A unique integer vector of snp IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getChromosome(object, index, char=FALSE)`: A vector of chromosomes. The optional `index` is a logical or integer vector specifying elements to extract. If `char=FALSE` (default), returns an integer vector. If `char=TRUE`, returns a character vector with elements in (1:22,X,XY,Y,M,U). "U" stands for "Unknown" and is the value given to any chromosome code not falling in the other categories.

`getPosition(object, index)`: An integer vector of base pair positions. The optional `index` is a logical or integer vector specifying elements to extract.

`getAlleleA(object, index)`: A character vector of A alleles. The optional `index` is a logical or integer vector specifying elements to extract.

`getAlleleB(object, index)`: A character vector of B alleles. The optional `index` is a logical or integer vector specifying elements to extract.

`getScanID(object, index)`: A unique integer vector of scan IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getGenotype(object, snp, scan, transpose=FALSE)`: Extracts genotype values (number of A alleles). The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`. Genotypes are returned in SNP x scan order if transpose=FALSE, otherwise they are returned in scan x SNP order.

`getVariable(object, varname, snp, scan)`: Extracts the contents of the variable `varname`. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`. If the variable is not found in the GDS file, returns `NULL`.

`XchromCode(object)`: Returns the integer code for the X chromosome.

`XYchromCode(object)`: Returns the integer code for the pseudoautosomal region.

`YchromCode(object)`: Returns the integer code for the Y chromosome.

`MchromCode(object)`: Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

`GdsReader`, `GenotypeData`

Examples

```
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)

# dimensions
nsnp(gds)
nscan(gds)
```

```
# get snpID and chromosome
snpID <- getSnpID(gds)
chrom <- getChromosome(gds)

# get positions only for chromosome 22
pos22 <- getPosition(gds, index=(chrom == 22))

# get all snps for first scan
geno <- getGenotype(gds, snp=c(1,-1), scan=c(1,1))

# starting at snp 100, get 10 snps for the first 5 scans
geno <- getGenotype(gds, snp=c(100,10), scan=c(1,5))

close(gds)
```

---

gdsImputedDosage *Create and check a GDS file with imputed dosages - deprecated*

Description

Deprecated - use imputedDosageFile, checkImputedDosageFile

These functions create or check a GDS file and corresponding annotation for imputed dosages from IMPUTE2, BEAGLE, or MaCH.

Usage

```
gdsImputedDosage(input.files, gds.filename, chromosome,
                  input.type=c("IMPUTE2", "BEAGLE", "MaCH"),
                  input.dosage=FALSE, block.size=5000,
                  snp.annot.filename="dosage.snp.RData",
                  scan.annot.filename="dosage.scan.RData",
                  verbose=TRUE, zipflag="ZIP.max",
                  genotypeDim="snp,scan",
                  scan.df=NULL, snp.exclude=NULL, snp.id.start=1)
gdsCheckImputedDosage(genoData, snpAnnot, scanAnnot,
                      input.files, chromosome,
                      input.type=c("IMPUTE2", "BEAGLE", "MaCH"),
                      input.dosage=FALSE, block.size=5000,
                      verbose=TRUE,
                      snp.exclude=NULL,
                      snp.id.start=1,
                      tolerance=1e-4,
                      na.logfile=NULL)
```

Arguments

input.files	A character vector of input files. The first file should always be genotypes (either probabilities or dosages). Files for each input type should be as follows:

- IMPUTE2: 1) .gens, 2) .samples
- BEAGLE: 1) .grobs or .dose, 2) .markers
- MaCH: 1) .mlprob or .mldose, 2) .mlinfo, 3) file with columns named "SNP" and "position" giving base pair position of all SNPs

`gds.filename` Character string with name of output GDS file.

`chromosome` Chromosome corresponding to the SNPs in the genotype file. Character codes will be mapped to integer values as follows: "X"->23, "XY"->24, "Y"-> 25, "M","MT"->26.

`input.type` Format of input files. Accepted file types are "IMPUTE2", "BEAGLE", and "MaCH".

`input.dosage` Logical for whether the genotype file (`input.files[1]`) contains dosages. If `FALSE` (default), the genotype file is assumed to contain genotype probabilities.

`block.size` Number of lines to read at once.

`snp.annot.filename`
Output .RData file for storing a `SnpAnnotationDataFrame`.

`scan.annot.filename`
Output .RData file for storing a `ScanAnnotationDataFrame`.

`verbose` Logical for whether to print progress messages.

`genotypeDim` character string specifying genotype dimensions of gds file. Either "snp,scan" or "scan,snp"

`zipflag` compression method for GDS nodes, can be "", "ZIP", "ZIP.fast", "ZIP.default", "ZIP.max"

`scan.df` data frame specifying which samples to include in the output GDS files, with optional scanIDs already assigned. See details.

`snp.exclude` vector of integers specifying which SNPs to exclude from the GDS file.

`snp.id.start` Starting index for snpID.

`genoData` A `GenotypeData` object from a GDS file created with `gdsImputedDosage`.

`snpAnnot` The `SnpAnnotationDataFrame` created by `gdsImputedDosage`

`scanAnnot` The `ScanAnnotationDataFrame` created by `gdsImputedDosage`

`tolerance` tolerance for checking differences against input files

`na.logfile` filename for recording snpID and scanID of missing dosages

Details

Input files can contain either imputed dosages or genotype probabilities, specified by the `input.dosage` flag. In either case, the GDS file will store dosage of the A allele in the "genotype" variable. All SNPs are assumed to be on the same chromosome, which is indicated by the `chromosome` argument.

If the input file contains genotype probabilities for all three genotypes, the dosage is set to missing if the genotype probability strings (before numerical conversion) are equal (e.g., (0,0,0), (0.33, 0.33, 0.33), or (-1, -1, -1)). The dosage is also normalized by the sum of all three genotype probabilities.

The `scan.df` argument allows the user to specify what samples should be included in the GDS files and an optional sampleID-scanID mapping. `scan.df` is a data frame with required column `sampleID`. The function attempts to match the given sampleID in the `scan.df` data frame with a unique sampleID in the input files. The format of sampleID is different for different input types:

- IMPUTE2: "ID_1 ID_2" as given in the sample file, where IDs are separated by a space
- BEAGLE: Column header names corresponding to that sample in .dose or .gprobs file
- MaCH: The first column of the .mlprob or .mlprob file

The `snp.names` argument allows the user to specify the which SNPs should be included in the GDS files. However, `snp.names` must be in the same order as SNPs occur in the imputation files; this option therefore only allows selection of SNPs, not reordering of SNPs. The ordering is checked and an error is thrown if the SNP names are not in order, but due to the design of imputation files, this may not occur until well into the GDS file population. The user can specify the starting snpID by setting `snp.id.start`, and included SNPs are numbered sequentially starting with `snp.id.start`. For IMPUTE2 data, snp.names must correspond to the second column of the .gprobs file.

Minimal SNP and scan annotation are created from the input files and stored in RData format in `snp.annot.filename` and `scan.annot.filename`.

If requested with `na.logfile`, `gdsCheckImputedDosage` will output a file with scanIDs and snpIDs of missing genotype calls.

Currently supported input file types are IMPUTE2, BEAGLE, and MaCH.

Author(s)

Adrienne Stilp, Stephanie Gogarten

References

IMPUTE2: http://mathgen.stats.ox.ac.uk/impute/impute_v2.html

BEAGLE: http://faculty.washington.edu/browning/beagle/beagle.html

MaCH: http://www.sph.umich.edu/csg/abecasis/MACH/tour/imputation.html

See Also

createDataFile, GdsGenotypeReader, GenotypeData, assocTestRegression

Examples

```
## Not run:
gdsfile <- tempfile()
snpfile <- tempfile()
scanfile <- tempfile()
logfile <- tempfile()

# IMPUTE2
probfile <- system.file("extdata", "imputation", "IMPUTE2", "example.chr22.study.gens",
                        package="GWASdata")
sampfile <- system.file("extdata", "imputation", "IMPUTE2", "example.study.samples",
                        package="GWASdata")
gdsImputedDosage(input.files=c(probfile, sampfile), gds.filename=gdsfile, chromosome=22,
                 input.type="IMPUTE2", input.dosage=FALSE,
                 snp.annot.filename=snpfile, scan.annot.filename=scanfile)

gds <- GdsGenotypeReader(gdsfile)
```

```
scanAnnot <- getobj(scanfile)
snpAnnot <- getobj(snpfile)
genoData <- GenotypeData(gds, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

gdsCheckImputedDosage(genoData, snpAnnot, scanAnnot,
                      input.files=c(probfile, sampfile), chromosome=22,
                      input.type="IMPUTE2", input.dosage=FALSE, na.logfile=logfile)

geno <- getGenotype(genoData)
getVariable(genoData, "alleleA")
getVariable(genoData, "alleleB")

log <- read.table(logfile)
head(log)

# association test with imputed dosages
scanAnnot$status <- sample(0:1, nrow(scanAnnot), replace=TRUE)
genoData <- GenotypeData(gds, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
assoc <- assocTestRegression(genoData, outcome="status", model.type="logistic",
                             gene.action.list="additive", dosage=TRUE)
head(assoc)
close(genoData)


# BEAGLE - genotype probabilities
probfile <- system.file("extdata", "imputation", "BEAGLE", "example.hapmap.unphased.bgl.gprobs",
                        package="GWASdata")
markfile <- system.file("extdata", "imputation", "BEAGLE", "hapmap.markers",
                        package="GWASdata")
gdsImputedDosage(input.files=c(probfile, markfile), gds.filename=gdsfile, chromosome=22,
                 input.type="BEAGLE", input.dosage=FALSE,
                 snp.annot.filename=snpfile, scan.annot.filename=scanfile)

# BEAGLE - dosage
dosefile <- system.file("extdata", "imputation", "BEAGLE", "example.hapmap.unphased.bgl.dose",
                        package="GWASdata")
gdsImputedDosage(input.files=c(dosefile, markfile), gds.filename=gdsfile, chromosome=22,
                 input.type="BEAGLE", input.dosage=TRUE,
                 snp.annot.filename=snpfile, scan.annot.filename=scanfile)


# MaCH - genotype probabilities
probfile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mlprob",
                        package="GWASdata")
markfile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mlinfo",
                        package="GWASdata")
posfile <- system.file("extdata", "imputation", "MaCH", "mach1.snp.position",
                        package="GWASdata")
gdsImputedDosage(input.files=c(probfile, markfile, posfile), gds.filename=gdsfile, chromosome=22,
                 input.type="MaCH", input.dosage=FALSE,
                 snp.annot.filename=snpfile, scan.annot.filename=scanfile)

# MaCH - dosage
```

```
dosefile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mldose",
                        package="GWASdata")
gdsImputedDosage(input.files=c(dosefile, markfile, posfile), gds.filename=gdsfile, chromosome=22,
                 input.type="MaCH", input.dosage=TRUE,
                 snp.annot.filename=snpfile, scan.annot.filename=scanfile)

unlink(c(gdsfile, snpfile, scanfile))

## End(Not run)
```

GdsIntensityReader *Class GdsIntensityReader*

Description

The GdsIntensityReader class is an extension of the GdsReader class specific to reading intensity data stored in GDS files.

Extends

GdsReader

Constructor

`GdsIntensityReader(filename)`:

`filename` must be the path to a GDS file. The GDS file must contain the following variables:

- 'snp': a coordinate variable with a unique integer vector of snp ids
- 'chromosome': integer chromosome values of dimension 'snp'
- 'position': integer position values of dimension 'snp'
- 'sampleID': a unique integer vector of scan ids with dimension 'sample'

Default values for chromosome codes are 1-22=autosome, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `autosomeCode`, `XchromCode`, `XYchromCode`, `YchromCode`, and `MchromCode`.

The GDS file should also contain at least one of the following variables with dimensions ('snp','sample'):

- 'quality': quality score
- 'X': X intensity
- 'Y': Y intensity
- 'BAlleleFreq': B allele frequency
- 'LogRRatio': Log R Ratio

The `GdsIntensityReader` constructor creates and returns a GdsIntensityReader instance pointing to this file.

Accessors

In the code snippets below, `object` is a GdsIntensityReader object. `snp` and `scan` indicate which elements to return along the snp and scan dimensions. They must be integer vectors of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that the entire dimension should be read. If snp and/or is scan omitted, the entire variable is read.

See `GdsReader` for additional methods.

`nsnp(object)`: The number of SNPs in the GDS file.

`nscan(object)`: The number of scans in the GDS file.

`getSnpID(object, index)`: A unique integer vector of snp IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getChromosome(object, index, char=FALSE)`: A vector of chromosomes. The optional `index` is a logical or integer vector specifying elements to extract. If `char=FALSE` (default), returns an integer vector. If `char=TRUE`, returns a character vector with elements in (1:22,X,XY,Y,M,U). "U" stands for "Unknown" and is the value given to any chromosome code not falling in the other categories.

`getPosition(object, index)`: An integer vector of base pair positions. The optional `index` is a logical or integer vector specifying elements to extract.

`getScanID(object, index)`: A unique integer vector of scan IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getQuality(object)`: Extracts quality scores. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the GDS file contains a variable 'quality'. `hasQuality(object)`:

`getX(object)`: Extracts X intensity. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the GDS file contains a variable 'X'. `hasX(object)`:

`getY(object)`: Extracts Y intensity. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the GDS file contains a variable 'Y'. `hasY(object)`:

`getBAlleleFreq(object)`: Extracts B allele frequency. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the GDS file contains a variable 'BAlleleFreq'. `hasBAlleleFreq(object)`:

`getLogRRatio(object)`: Extracts Log R Ratio. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the GDS file contains a variable 'LogRRatio'. `hasLogRRatio(object)`:

`getVariable(object, varname, snp, scan)`: Returns the contents of the variable `varname`. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`. If the variable is not found in the GDS file, returns `NULL`.

`autosomeCode(object)`: Returns the integer codes for the autosomes.

`XchromCode(object)`: Returns the integer code for the X chromosome.

`XYchromCode(object)`: Returns the integer code for the pseudoautosomal region.

`YchromCode(object)`: Returns the integer code for the Y chromosome.

`McchromCode(object)`: Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

`GdsReader`, `GdsGenotypeReader`, `GenotypeData`, `IntensityData`

Examples

```
file <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
gds <- GdsIntensityReader(file)

# dimensions
nsnp(gds)
nscan(gds)

# get snpID and chromosome
snpID <- getSnpID(gds)
chrom <- getChromosome(gds)

# get positions only for chromosome 22
pos22 <- getPosition(gds, index=(chrom == 22))

# get all snps for first scan
x <- getX(gds, snp=c(1,-1), scan=c(1,1))

# starting at snp 100, get 10 snps for the first 5 scans
x <- getX(gds, snp=c(100,10), scan=c(1,5))

close(gds)
```

GdsReader *Class GdsReader*

Description

The GdsReader class provides an interface for reading GDS files.

Constructor

`GdsReader(filename, ...)`:

`filename` must be the path to a GDS file or an already opened gds object.

The `GdsReader` constructor creates and returns a GdsReader instance pointing to this file.

Accessors

In the code snippets below, `object` is a GdsReader object.

`getVariable(object, varname, start, count)`: Returns the contents of the variable `varname`.

- `start` is a vector of integers indicating where to start reading values. The length of this vector must equal the number of dimensions the variable has. If not specified, reading starts at the beginning of the file (1,1,...).
- `count` is a vector of integers indicating the count of values to read along each dimension. The length of this vector must equal the number of dimensions the variable has. If not specified and the variable does NOT have an unlimited dimension, the entire variable is read. As a special case, the value "-1" indicates that all entries along that dimension should be read.

The result is a vector, matrix, or array, depending on the number of dimensions in the returned values. Missing values (specified by a "missing.value" attribute, see `put.attr.gdsn`) are represented as `NA`. If the variable is not found in the GDS file, returns `NULL`.

`getVariableNames(object)`: Returns names of variables in the GDS file.

`getDimension(object, varname)`: Returns dimension for GDS variable `varname`.

`getNodeDescription(object, varname)`: Returns description for GDS variable `varname`.

`getAttribute(object, attname, varname)`: Returns the attribute `attname` associated with the variable `varname`.

`hasVariable(object, varname)`: Returns `TRUE` if `varname` is a variable in the GDS file.

Standard Generic Methods

In the code snippets below, `object` is a GdsReader object.

`open(object)`: Opens a connection to a GDS file.

`close(object)`: Closes the connection to a GDS file.

`show(object)`: Summarizes the contents of a GDS file.

Author(s)

Stephanie Gogarten

See Also

gdsfmt

Examples

```
library(SNPRelate)
gds <- GdsReader(snpgdsExampleFileName())

getVariableNames(gds)

hasVariable(gds, "genotype")
geno <- getVariable(gds, "genotype", start=c(1,1), count=c(10,10))
```

```
close(gds)
```

---

gdsSubset *Write a subset of data in a GDS file to a new GDS file*

Description

gdsSubset takes a subset of data (snps and samples) from a GDS file and write it to a new GDS file. gdsSubsetCheck checks that a GDS file is the desired subset of another GDS file.

Usage

```
gdsSubset(parent.gds, sub.gds,
          sample.include=NULL, snp.include=NULL,
          sub.storage=NULL,
          compress="ZIP.max",
          block.size=5000,
          verbose=TRUE)

gdsSubsetCheck(parent.gds, sub.gds,
               sample.include=NULL, snp.include=NULL,
               sub.storage=NULL,
               verbose=TRUE)
```

Arguments

parent.gds	Name of the parent GDS file
sub.gds	Name of the subset GDS file
sample.include	Vector of sampleIDs to include in sub.gds
snp.include	Vector of snpIDs to include in sub.gds
sub.storage	storage type for the subset file; defaults to original storage type
compress	compression for GDS file, can be "", "ZIP", "ZIP.fast", "ZIP.default", "ZIP.max"
block.size	for GDS files stored with scan,snp dimensions, the number of SNPs to read from the parent file at a time. Ignored for snp,scan dimensions.
verbose	Logical value specifying whether to show progress information.

Details

gdsSubset can select a subset of snps for all samples by setting snp.include, a subset of samples for all snps by setting sample.include, or a subset of snps and samples with both arguments. The GDS nodes "snp.id", "snp.position", "snp.chromosome", and "sample.id" are copied, as well as any 2-dimensional nodes. Other nodes are not copied. The attributes of the 2-dimensional nodes are also copied to the subset file. If sub.storage is specified, the subset gds file will have a different storage mode for any 2-dimensional array. In the special case where the 2-dimensional node has an

attribute named "missing.value" and the sub.storage type is "bit2", the missing.value attribute for the subset node is automatically set to 3. At this point, no checking is done to ensure that the values will be properly stored with a different storage type, but gdsSubsetCheck will return an error if the values do not match. If the nodes in the GDS file are stored with scan,snp dimensions, then block.size allows you to loop over a block of SNPs at a time. If the nodes are stored with snp,scan dimensions, then the function simply loops over samples, one at a time.

gdsSubsetCheck checks that a subset GDS file has the expected SNPs and samples of the parent file. It also checks that attributes were similarly copied, except for the above-mentioned special case of missing.value for sub.storage="bit2".

Author(s)

Adrienne Stilp

See Also

gdsfmt, createDataFile

Examples

```
gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(gdsfile)
sample.sel <- getScanID(gds, index=1:10)
snp.sel <- getSnpID(gds, index=1:100)
close(gds)

subfile <- tempfile()
gdsSubset(gdsfile, subfile, sample.include=sample.sel, snp.include=snp.sel)
gdsSubsetCheck(gdsfile, subfile, sample.include=sample.sel, snp.include=snp.sel)

file.remove(subfile)
```

---

genoClusterPlot *SNP cluster plots*

Description

Generates either X,Y or R,Theta cluster plots for specified SNP's.

Usage

```
genoClusterPlot(intenData, genoData, plot.type = c("RTheta", "XY"),
                snpID, main.txt = NULL, by.sex = FALSE,
                scan.sel = NULL, scan.hilite = NULL,
                start.axis.at.0 = FALSE,
                verbose = TRUE, ...)

genoClusterPlotByBatch(intenData, genoData, plot.type = c("RTheta", "XY"),
                       snpID, batchVar, main.txt = NULL, scan.sel = NULL,
                       verbose = TRUE, ...)
```

Arguments

intenData	IntensityData object containing 'X' and 'Y' values.
genoData	GenotypeData object
plot.type	The type of plots to generate. Possible values are "RTheta" (default) or "XY".
snpID	A numerical vector containing the SNP number for each plot.
batchVar	A character string indicating which annotation variable should be used as the batch.
main.txt	A character vector containing the title to give to each plot.
by.sex	Logical value specifying whether to indicate sex on the plot. If TRUE, sex must be present in intenData or genoData.
scan.sel	integer vector of scans to include in the plot. If NULL, all scans will be included.
scan.hilite	integer vector of scans to highlight in the plot with different colors. If NULL, all scans will be plotted with the same colors.
start.axis.at.0	Logical for whether the min value of each axis should be 0.
verbose	Logical value specifying whether to show progress.
...	Other parameters to be passed directly to plot.

Details

Either 'RTheta' (default) or 'XY' plots can be generated. R and Theta values are computed from X and Y using the formulas `r <- x+y` and `theta <- atan(y/x)*(2/pi)`.

If `by.sex==TRUE`, females are indicated with circles and males with crosses.

Author(s)

Caitlin McHugh

See Also

IntensityData, GenotypeData

Examples

```
# create data object
library(GWASdata)
data(illuminaScanADF, illuminaSnpADF)

xyfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
xy <- GdsIntensityReader(xyfile)
xyData <-  IntensityData(xy, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <-  GenotypeData(geno, scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)
```

```
# select first 9 snps
snpID <- illuminaSnpADF$snpID[1:9]
rsID <- illuminaSnpADF$rsID[1:9]

par(mfrow=c(3,3)) # plot 3x3
genoClusterPlot(xyData, genoData, snpID=snpID, main.txt=rsID)

# select samples
scan.sel <- illuminaScanADF$scanID[illuminaScanADF$race == "CEU"]
genoClusterPlot(xyData, genoData, snpID=snpID, main.txt=rsID,
                scan.sel=scan.sel, by.sex=TRUE)

genoClusterPlot(xyData, genoData, snpID=snpID, main.txt=rsID,
                scan.hilite=scan.sel)

genoClusterPlotByBatch(xyData, genoData, snpID=snpID, main.txt=rsID,
                       batchVar="plate")

close(xyData)
close(genoData)
```

`GenotypeData-class` *Class GenotypeData*

Description

The GenotypeData class is a container for storing genotype data from a genome-wide association study together with the metadata associated with the subjects and SNPs involved in the study.

Details

The GenotypeData class consists of three slots: data, snp annotation, and scan annotation. There may be multiple scans associated with a subject (e.g. duplicate scans for quality control), hence the use of "scan" as one dimension of the data. Snp and scan annotation are optional, but if included in the GenotypeData object, their unique integer ids (snpID and scanID) are checked against the ids stored in the data slot to ensure consistency.

Constructor

`GenotypeData(data, snpAnnot=NULL, scanAnnot=NULL)`:

`data` must be an `NcdfGenotypeReader`, `GdsGenotypeReader`, or `MatrixGenotypeReader` object.

`snpAnnot`, if not `NULL`, must be a `SnpAnnotationDataFrame` or `SnpAnnotationSQLite` object.

`scanAnnot`, if not `NULL`, must be a `ScanAnnotationDataFrame` or `ScanAnnotationSQLite` object.

The `GenotypeData` constructor creates and returns a GenotypeData instance, ensuring that data, snpAnnot, and scanAnnot are internally consistent.

Accessors

In the code snippets below, `object` is a GenotypeData object. `snp` and `scan` indicate which elements to return along the snp and scan dimensions. They must be integer vectors of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that the entire dimension should be read. If snp and/or is scan omitted, the entire variable is read.

`nsnp(object)`: The number of SNPs in the data.

`nscan(object)`: The number of scans in the data.

`getSnpID(object, index)`: A unique integer vector of snp IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getChromosome(object, index, char=FALSE)`: A vector of chromosomes. The optional `index` is a logical or integer vector specifying elements to extract. If `char=FALSE` (default), returns an integer vector. If `char=TRUE`, returns a character vector with elements in (1:22,X,XY,Y,M,U).

`getPosition(object, index)`: An integer vector of base pair positions. The optional `index` is a logical or integer vector specifying elements to extract.

`getAlleleA(object, index)`: A character vector of A alleles. The optional `index` is a logical or integer vector specifying elements to extract.

`getAlleleB(object, index)`: A character vector of B alleles. The optional `index` is a logical or integer vector specifying elements to extract.

`getScanID(object, index)`: A unique integer vector of scan IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getSex(object, index)`: A character vector of sex, with values 'M' or 'F'. The optional `index` is a logical or integer vector specifying elements to extract.

`hasSex(object)`: Returns `TRUE` if the column 'sex' is present in `object`.

`getGenotype(object, snp, scan, char=FALSE, sort=TRUE)`: Extracts genotype values (number of A alleles). The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`. If `char=TRUE`, genotypes are returned as characters of the form "A/B". If `sort=TRUE`, alleles are lexographically sorted ("G/T" instead of "T/G").

`getSnpVariable(object, varname, index)`: Returns the snp annotation variable `varname`. The optional `index` is a logical or integer vector specifying elements to extract.

`getSnpVariableNames(object)`: Returns a character vector with the names of the columns in the snp annotation.

`hasSnpVariable(object, varname)`: Returns `TRUE` if the variable `varname` is present in the snp annotation.

`getScanVariable(object, varname, index)`: Returns the scan annotation variable `varname`. The optional `index` is a logical or integer vector specifying elements to extract.

`getScanVariableNames(object)`: Returns a character vector with the names of the columns in the scan annotation.

`hasScanVariable(object, varname)`: Returns `TRUE` if the variable `varname` is present in the scan annotation.

`getVariable(object, varname, snp, scan)`: Extracts the contents of the variable `varname` from the data. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`. If the variable is not found, returns `NULL`.

`hasVariable(object, varname)`: Returns `TRUE` if the data contains contains `varname`, `FALSE` if not.

`hasSnpAnnotation(object)`: Returns `TRUE` if the snp annotation slot is not `NULL`.

`hasScanAnnotation(object)`: Returns `TRUE` if the scan annotation slot is not `NULL`.

`open(object)`: Opens a connection to the data.

`close(object)`: Closes the data connection.

`autosomeCode(object)`: Returns the integer codes for the autosomes.

`XchromCode(object)`: Returns the integer code for the X chromosome.

`XYchromCode(object)`: Returns the integer code for the pseudoautosomal region.

`YchromCode(object)`: Returns the integer code for the Y chromosome.

`McchromCode(object)`: Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

SnpAnnotationDataFrame, SnpAnnotationSQLite, ScanAnnotationDataFrame, ScanAnnotationSQLite, GdsGenotypeReader, NcdfGenotypeReader, MatrixGenotypeReader, IntensityData

Examples

```
library(GWASdata)
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)

# object without annotation
genoData <- GenotypeData(gds)

# object with annotation
data(illuminaSnpADF)
data(illuminaScanADF)
# need to rebuild old SNP annotation object to get allele methods
snpAnnot <- SnpAnnotationDataFrame(pData(illuminaSnpADF))
genoData <-  GenotypeData(gds, snpAnnot=snpAnnot, scanAnnot=illuminaScanADF)

# dimensions
nsnp(genoData)
nscan(genoData)

# get snpID and chromosome
snpID <- getSnpID(genoData)
chrom <- getChromosome(genoData)
```

```
# get positions only for chromosome 22
pos22 <- getPosition(genoData, index=(chrom == 22))

# get other annotations
if (hasSex(genoData)) sex <- getSex(genoData)
plate <- getScanVariable(genoData, "plate")
rsID <- getSnpVariable(genoData, "rsID")

# get all snps for first scan
geno <- getGenotype(genoData, snp=c(1,-1), scan=c(1,1))

# starting at snp 100, get 10 snps for the first 5 scans
geno <- getGenotype(genoData, snp=c(100,10), scan=c(1,5))
geno

# return genotypes as "A/B" rather than number of A alleles
geno <- getGenotype(genoData, snp=c(100,10), scan=c(1,5), char=TRUE)
geno

close(genoData)

#--------------------------------------
# An example using a non-human organism
#--------------------------------------
# Chicken has 38 autosomes, Z, and W.  Male is ZZ, female is ZW.
# Define sex chromosomes as X=Z and Y=W.
ncfile <- tempfile()
simulateGenotypeMatrix(n.snps=10, n.chromosomes=40, n.samples=5,
                       ncdf.filename=ncfile)
nc <- NcdfGenotypeReader(ncfile, autosomeCode=1:38L,
                         XchromCode=39L, YchromCode=40L,
                         XYchromCode=41L, MchromCode=42L)
table(getChromosome(nc))
table(getChromosome(nc, char=TRUE))

# SNP annotation
snpdf <- data.frame(snpID=getSnpID(nc),
                    chromosome=getChromosome(nc),
                    position=getPosition(nc))
snpAnnot <- SnpAnnotationDataFrame(snpdf, autosomeCode=1:38L,
                         XchromCode=39L, YchromCode=40L,
                         XYchromCode=41L, MchromCode=42L)
varMetadata(snpAnnot)[,"labelDescription"] <-
  c("unique integer ID",
    "chromosome coded as 1:38=autosomes, 39=Z, 40=W",
    "base position")

# reverse sex coding to get proper counting of sex chromosome SNPs
scandf <- data.frame(scanID=1:5, sex=c("M","M","F","F","F"),
                     stringsAsFactors=FALSE)
scanAnnot <- ScanAnnotationDataFrame(scandf)
varMetadata(scanAnnot)[,"labelDescription"] <-
```

```
  c("unique integer ID",
    "sex coded as M=female and F=male")

genoData <- GenotypeData(nc, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
afreq <- alleleFrequency(genoData)
# frequency of Z chromosome in females ("M") and males ("F")
afreq[snpAnnot$chromosome == 39, c("M","F")]
# frequency of W chromosome in females ("M") and males ("F")
afreq[snpAnnot$chromosome == 40, c("M","F")]

close(genoData)
unlink(ncfile)
```

---

genotypeToCharacter *Convert number of A alleles to character genotypes*

Description

Converts a vector or matrix of genotypes encoded as number of A alleles to character strings of the form "A/B".

Usage

```
genotypeToCharacter(geno, alleleA=NULL, alleleB=NULL, sort=TRUE)
```

Arguments

geno	Vector or matrix of genotype values, encoded as number of A alleles. If a matrix, dimensions should be (snp, sample).
alleleA	Character vector with allele A.
alleleB	Character vector with allele B.
sort	Logical for whether to sort alleles lexographically ("G/T" instead of "T/G").

Details

If geno is a vector, alleleA and alleleB should have the same length as geno or length 1 (in the latter case the values are recycled).

If geno is a matrix, length of alleleA and alleleB should be equal to the number of rows of geno.

If alelleA or alleleB is NULL, returned genotypes will have values "A/A", "A/B", or "B/B".

Value

Character vector or matrix of the same dimensions as geno.

Author(s)

Stephanie Gogarten

See Also

GenotypeData

Examples

```
geno <- matrix(c(0,1,2,0,1,2,1,NA), nrow=4)
alleleA <- c("A","T","C","T")
alleleB <- c("C","G","G","A")
genotypeToCharacter(geno, alleleA, alleleB)
```

---

getobj *Get an R object stored in an Rdata file*

Description

Returns an R object stored in an Rdata file

Usage

```
getobj(Rdata)
```

Arguments

Rdata	path to an Rdata file containing a single R object to load

Details

Loads an R object and stores it under a new name without creating a duplicate copy. If multiple objects are stored in the same file, only the first one will be returned

Value

The R object stored in Rdata.

Author(s)

Stephanie Gogarten

See Also

saveas

Examples

```
x <- 1:10
file <- tempfile()
save(x, file=file)
y <- getobj(file)
unlink(file)
```

getVariable *Accessors for variables in GenotypeData and IntensityData classes and their component classes*

Description

These generic functions provide access to variables associated with GWAS data cleaning.

Usage

```
getScanVariable(object, varname, ...)
getScanVariableNames(object, ...)
getScanID(object, ...)
getSex(object, ...)
getSnpVariable(object, varname, ...)
getSnpVariableNames(object, ...)
getSnpID(object, ...)
getChromosome(object, ...)
getPosition(object, ...)
getAlleleA(object, ...)
getAlleleB(object, ...)

getVariable(object, varname, ...)
getVariableNames(object, ...)
getGenotype(object, ...)
getQuality(object, ...)
getX(object, ...)
getY(object, ...)
getBAlleleFreq(object, ...)
getLogRRatio(object, ...)
getDimension(object, varname, ...)
getAttribute(object, attname, varname, ...)
getNodeDescription(object, varname, ...)

getAnnotation(object, ...)
getMetadata(object, ...)
getQuery(object, statement, ...)

hasScanAnnotation(object)
hasScanVariable(object, varname)
hasSex(object)
hasSnpAnnotation(object)
hasSnpVariable(object, varname)
hasVariable(object, varname)
hasQuality(object)
hasX(object)
hasY(object)
```

```
hasBAlleleFreq(object)
hasLogRRatio(object)

nsnp(object)
nscan(object)

autosomeCode(object)
XchromCode(object)
XYchromCode(object)
YchromCode(object)
MchromCode(object)

writeAnnotation(object, value, ...)
writeMetadata(object, value, ...)
```

Arguments

object	Object, possibly derived from or containing NcdfReader-class, GdsReader-class, ScanAnnotationDataFrame-class, SnpAnnotationDataFrame-class, ScanAnnotationSQLite-clas or SnpAnnotationSQLite-class.
varname	Name of the variable (single character string, or a character vector for multiple variables).
attname	Name of an attribute.
statement	SQL statement to query ScanAnnotationSQLite-class or SnpAnnotationSQLite-class objects.
value	data.frame with annotation or metadata to write to ScanAnnotationSQLite-class or SnpAnnotationSQLite-class objects.
...	Additional arguments.

Value

get methods return vectors or matrices of the requested variables (with the exception of getQuery, which returns a data frame).

has methods return TRUE if the requested variable is present in object.

nsnp and nscan return the number of SNPs and scans in the object, repectively.

autosomeCode, XchromCode, XYchromCode, YchromCode, and MchromCode return the integer chromosome codes associated with autosomal, X, pseudoautosomal, Y, and mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

ScanAnnotationDataFrame-class, SnpAnnotationDataFrame-class, ScanAnnotationSQLite-class, SnpAnnotationSQLite-class, NcdfReader-class, NcdfGenotypeReader-class, NcdfIntensityReader-class, GdsReader-class, GenotypeData-class, IntensityData-class

gwasExactHW *Hardy-Weinberg Equilibrium testing*

Description

This function performs exact Hardy-Weinberg Equilibrium testing (using Fisher's Test) over a selection of SNPs. It also performs genotype counts, calculates allele frequencies, and calculates inbreeding coefficients.

Usage

```
gwasExactHW(genoData,
    scan.chromosome.filter = NULL,
    scan.exclude = NULL,
    geno.counts = TRUE,
    chromosome.set = NULL,
    block.size = 5000,
    verbose = TRUE,
    outfile = NULL)
```

Arguments

genoData	GenotypeData object, should contain sex and phenotypes in scan annotation. Chromosomes are expected to be in contiguous blocks.
scan.chromosome.filter	a logical matrix that can be used to exclude some chromosomes, some scans, or some specific scan-chromosome pairs. Entries should be TRUE if that scan-chromosome pair should be included in the analysis, FALSE if not. The number of rows must be equal to the number of scans in genoData, and the number of columns must be equal to the largest integer chromosome value in genoData. The column number must match the chromosome number. e.g. A scan.chromosome.filter matrix used for an analyis when genoData has SNPs with chromosome=(1-24, 26, 27) (i.e. no Y (25) chromosome SNPs) must have 27 columns (all FALSE in the 25th column). But a scan.chromosome.filter matrix used for an analysis genoData has SNPs chromosome=(1-26) (i.e no Unmapped (27) chromosome SNPs) must have only 26 columns.
scan.exclude	an integer vector containing the IDs of entire scans to be excluded.
geno.counts	if TRUE (default), genotype counts are returned in the output data.frame.
chromosome.set	integer vector with chromosome(s) to be analyzed. Use 23, 24, 25, 26, 27 for X, XY, Y, M, Unmapped respectively.
block.size	Number of SNPs to be read from genoData at once.
verbose	if TRUE (default), will print status updates while the function runs. e.g. it will print "chr 1 block 1 of 10" etc. in the R console after each block of SNPs is done being analyzed.

`outfile` a character string to append in front of ".chr.i_k.RData" for naming the output data-frame; where i is the first chromosome, and k is the last chromosome used in that call to the function. "chr.i_k." will be omitted if `chromosome.set=NULL`.

Details

HWE calculations are performed with the `HWExact` function in the **GWASExactHW** package.

For the X chromosome, only female samples will be used in all calculations (since males are excluded from HWE testing on this chromosome). Hence if chromosome.set includes 23, the scan annotation of genoData should provide the sex of the sample ("M" or "F") i.e. there should be a column named "sex" with "F" for females and "M" for males.

Y, M, and U (25, 26, and 27) chromsome SNPs are not used in HWE analysis, so all returned values for these SNPs will be `NA`.

Value

If `outfile=NULL` (default), all results are returned as a single data.frame. If `outfile` is specified, no data is returned but the function saves a data-frame with the naming convention as described by the argument `outfile`.

The first three columns of the data-frame are:

`snpID` snpID (from the snp annotation) of the SNP

`chromosome` chromosome (from the snp annotation) of the SNP. The integers 23, 24, 25, 26, 27 are used for X, XY, Y, M, Unmapped respectively.

`position` position (from the snp annotation) of the SNP

If `geno.counts = TRUE`:

`nAA` number of AA genotypes in samples

`nAB` number of AB genotypes in samples

`nBB` number of BB genotypes in samples

`MAF` minor allele frequency.

`minor.allele` the minor allele. Takes values "A" or "B".

`f` the inbreeding coefficient.

`p.value` exact Hardy-Weinberg Equilibrium (using Fisher's Test) p-value. `p.value` will be `NA` for monomorphic SNPs (`MAF == 0`).

Warnings:

If `outfile` is not `NULL`, another file will be saved with the name "outfile.chr.i_k.warnings.RData" that contains any warnings generated by the function.

Author(s)

Ian Painter, Matthew P. Conomos

See Also

`HWExact`

Examples

```
# The following example would perform exact Hardy-Weinberg equilibrium testing on all chromosomes in this data set.

library(GWASdata)
data(illuminaScanADF)

# run only on YRI subjects
scan.exclude <- illuminaScanADF$scanID[illuminaScanADF$race != "YRI"]

# create data object
gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(gdsfile)
genoData <-  GenotypeData(gds, scanAnnot=illuminaScanADF)

hwe <- gwasExactHW(genoData, scan.exclude=scan.exclude)

close(genoData)
```

GWASTools-defunct *Defunct Functions in Package GWASTools*

Description

These functions are no longer part of GWASTools as they are no longer needed.

Details

`pedigreeClean` has been replaced by `pedigreeCheck`.

`pedigreeFindDuplicates` has been replaced by `pedigreeCheck`.

GWASTools-deprecated *Deprecated functions in package 'GWASTools'*

Description

These functions are provided for compatibility with older versions of 'GWASTools' only, and will be defunct at the next release.

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- ncdfCreate: `createDataFile`
- ncdfAddData: `createDataFile`
- ncdfAddIntensityData: `createAffyIntensityFile`

- ncdfCheckGenotype: checkGenotypeFile
- ncdfCheckIntensity: checkIntensityFile
- ncdfSetMissingGenotypes: setMissingGenotypes
- gdsSetMissingGenotypes: setMissingGenotypes
- ncdfImputedDosage: imputedDosageFile
- gdsImputedDosage: imputedDosageFile
- gdsCheckImputedDosage: checkImputedDosageFile

---

hetByScanChrom *Heterozygosity rates by scan and chromosome*

Description

This function calculates the fraction of heterozygous genotypes for each chromosome for a set of scans.

Usage

```
hetByScanChrom(genoData, snp.exclude = NULL,
               verbose = TRUE)
```

Arguments

genoData	GenotypeData object. Chromosomes are expected to be in contiguous blocks.
snp.exclude	An integer vector containing the id's of SNPs to be excluded.
verbose	Logical value specifying whether to show progress information.

Details

This function calculates the percent of heterozygous and missing genotypes in each chromosome of each scan given in genoData.

Value

The result is a matrix containing the heterozygosity rates with scans as rows and chromosomes as columns, including a column "A" for all autosomes.

Author(s)

Cathy Laurie

See Also

GenotypeData, hetBySnpSex

Examples

```
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)
genoData <- GenotypeData(gds)
het <- hetByScanChrom(genoData)
close(genoData)
```

---

hetBySnpSex *Heterozygosity by SNP and sex*

Description

This function calculates the percent of heterozygous genotypes for males and females for each SNP.

Usage

```
hetBySnpSex(genoData, scan.exclude = NULL,
            verbose = TRUE)
```

Arguments

genoData	GenotypeData object
scan.exclude	An integer vector containing the id's of scans to be excluded.
verbose	Logical value specifying whether to show progress information.

Details

This function calculates the percent of heterozygous genotypes for males and females for each SNP given in genoData. A "sex" variable must be present in the scan annotation slot of genoData.

Value

The result is a matrix containing the heterozygosity rates with snps as rows and 2 columns ("M" for males and "F" for females).

Author(s)

Cathy Laurie

See Also

GenotypeData, hetByScanChrom

Examples

```
library(GWASdata)
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)

# need scan annotation with sex
data(illuminaScanADF)
genoData <-  GenotypeData(gds, scanAnnot=illuminaScanADF)

het <- hetBySnpSex(genoData)
close(genoData)
```

HLA *HLA region base positions*

Description

HLA region base positions from the GRCh36/hg18 and GRCh37/hg19 genome builds.

Usage

```
HLA.hg18
HLA.hg19
```

Format

A data.frame with the following columns.

`chrom` chromsome

`start.base` starting base position of region

`end.base` ending base position of region

Source

UCSC genome browser (http://genome.ucsc.edu).

References

Mehra, Narinder K. and Kaur, Gurvinder (2003), MHC-based vaccination approaches: progress and perspectives. Expert Reviews in Molecular Medicine, Vol. 5: 24. doi:10.1017/S1462399403005957

Examples

```
data(HLA.hg18)
data(HLA.hg19)
```

ibdPlot *Plot theoretical and observed identity by descent values and assign relationships*

Description

ibdPlot produces an IBD plot showing observed identity by descent values color coded by expected relationship. Theoretical boundaries for full-sibling, second-degree, and third-degree relatives are plotted in orange. ibdAreasDraw overlays relationship areas for IBD analysis on the plot. ibdAssignRelatedness identifies observed relatives. ibdAssignRelatedness identifies observed relatives using the kinship coefficients and IBS0 estimates from the KING model.

Usage

```
ibdPlot(k0, k1, alpha=0.05, relation=NULL, color=NULL,
        rel.lwd=2, rel.draw=c("FS", "Deg2", "Deg3"), ...)

ibdAreasDraw(alpha=0.05, m=0.04, po.w=0.1, po.h=0.1,
             dup.w=0.1, dup.h=0.1, un.w=0.25, un.h=0.25, rel.lwd=2,
     xcol=c("cyan","red","blue","lightgreen","magenta","black"))

ibdAssignRelatedness(k0, k1, alpha=0.05, m=0.04, po.w=0.1, po.h=0.1,
                     dup.w=0.1, dup.h=0.1, un.w=0.25, un.h=0.25)

ibdAssignRelatednessKing(ibs0, kc, cut.kc.dup=1/(2^(3/2)),
                         cut.kc.fs=1/(2^(5/2)), cut.kc.deg2=1/(2^(7/2)),
                         cut.kc.deg3=1/(2^(9/2)), cut.ibs0.err=0.003)
```

Arguments

k0	A vector of k0 values.
k1	A vector of k1 values.
kc	A vector of kinship coefficient values (KING model).
ibs0	A vector of IBS0 values (KING model).
alpha	significance level - finds 100(1-alpha)% prediction intervals for second and third degree relatives and 100(1-alpha)% prediction ellipse for full siblings.
relation	A vector of relationships. Recognized values are "PO"=parent/offspring, "FS"=full siblings, "HS"=half siblings, "Av"=avuncular, "GpGc"=grandparent-grandchild, "Deg2"=any second-degree, "FC"=first cousins, "HAv"=half-avuncular, "Deg3"=any third degree, "U"=unrelated, and "Q"=unknown.
color	A vector of colors for (k0,k1) points.
rel.lwd	Line width for theoretical full-sib, Deg2, and Deg3 boundaries.
rel.draw	Which theoretical boundaries to plot: one or more of "FS" (full-sib), "Deg2" (second-degree), "Deg3" (third-degree). If NULL, no boundaries are drawn.

...	Other graphical parameters to pass to plot and points.
m	width of rectangle along diagonal line
po.w	width of parent-offspring rectangle
po.h	height of parent-offspring rectangle
dup.w	width of duplicate rectangle
dup.h	height of duplicate rectangle
un.w	width of unrelated rectangle
un.h	height of unrelated rectangle
xcol	colors for parent-offspring, full-sib, Deg2, Deg3, dup & unrelated areas
cut.kc.dup	Kinship coefficient threshold for dividing duplicates and first degree relatives.
cut.kc.fs	Kinship coefficient threshold for dividing full siblings and second degree relatives.
cut.kc.deg2	Kinship coefficient threshold for dividing second and third degree relatives.
cut.kc.deg3	Kinship coefficient threshold for dividing third degree relatives and unrelated.
cut.ibs0.err	IBS0 threshold for dividing parent-offsprings pairs from other relatives. Should be 0, but is usually slightly higher due to genotyping errors.

Details

ibdPlot produces an IBD plot showing observed identity by descent values color coded by expected relationship, typically as deduced from pedigree data. Points are plotted according to their corresponding value in the color vector, and the relation vector is used to make the plot legend. In addition to the relationships listed above, any relationships output from pedigreePairwiseRelatedness will be recognized.

Theoretical boundary for full-sibs is indicated by ellipse and boundaries for second and third degree intervals are indicated in orange. For full-sibs, 100(1-alpha)% prediction ellipse is based on assuming bivariate normal distribution with known mean and covariance matrix. For second degree (half siblings, avuncular, grandparent-grandchild) and third degree (first cousins), 100(1-alpha)% prediction intervals for k1 are based on assuming normal distribution with known mean and variance, computed as in Hill and Weir (2011).

ibdAreasDraw overlays relationship areas on the plot to help with analyzing observed relationships.

ibdAssignRelatedness identifies relatives based on their (k0, k1) coordinates.

ibdAssignRelatednessKing identifies relatives based on their (ibs0, kc) coordinates (KING model).

Value

ibdAssignRelatedness and ibdAssignRelatednessKing return a vector of relationships with values "Dup"=duplicate, "PO"=parent-offspring, "FS"=full sibling, "Deg2"=second degree, "Deg3"=third degree, "U"=unrelated, and "Q"=unknown.

Author(s)

Cathy Laurie, Cecelia Laurie, and Adrienne Stilp

References

Hill, W.G. and B.S. Weir, Variation in actual relationship as a consequence of mendelian sampling and linkage, Genet. Res. Camb. (2011), 93, 47-64.

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

See Also

relationsMeanVar, pedigreePairwiseRelatedness

Examples

```
k0 <- c(0, 0, 0.25, 0.5, 0.75, 1)
k1 <- c(0, 1, 0.5, 0.5, 0.25, 0)
exp.rel <- c("Dup", "PO", "FS", "HS", "FC", "U")
ibdPlot(k0, k1, relation=exp.rel)
ibdAreasDraw()
obs.rel <- ibdAssignRelatedness(k0, k1)

kc <- c(0.5, 0.25, 0.25, 0.125, 0.063, 0)
ibs0 <- c(0, 0, 0.25, 0.5, 0.75, 1)
obs.rel.king <- ibdAssignRelatednessKing(ibs0, kc)
```

---

imputedDosageFile *Create and check a GDS or NetCDF file with imputed dosages*

Description

These functions create or check a GDS or NetCDF file and corresponding annotation for imputed dosages from IMPUTE2, BEAGLE, or MaCH.

Usage

```
imputedDosageFile(input.files, filename, chromosome,
                  input.type=c("IMPUTE2", "BEAGLE", "MaCH"),
                  input.dosage=FALSE, file.type=c("gds", "ncdf"),
                  snp.annot.filename="dosage.snp.RData",
                  scan.annot.filename="dosage.scan.RData",
                  precision="single", compress="ZIP.max",
                  genotypeDim="snp,scan",
                  scan.df=NULL, snp.exclude=NULL, snp.id.start=1,
                  block.size=5000, verbose=TRUE)
checkImputedDosageFile(genoData, snpAnnot, scanAnnot,
                       input.files, chromosome,
                       input.type=c("IMPUTE2", "BEAGLE", "MaCH"),
                       input.dosage=FALSE,
                       snp.exclude=NULL, snp.id.start=1,
                       tolerance=1e-4, na.logfile=NULL,
                       block.size=5000, verbose=TRUE)
```

Arguments

input.files	A character vector of input files. The first file should always be genotypes (either probabilities or dosages). Files for each input type should be as follows: • IMPUTE2: 1) .gens, 2) .samples • BEAGLE: 1) .grobs or .dose, 2) .markers • MaCH: 1) .mlprob or .mldose, 2) .mlinfo, 3) file with columns named "SNP" and "position" giving base pair position of all SNPs
filename	Character string with name of output GDS or NetCDF file.
chromosome	Chromosome corresponding to the SNPs in the genotype file. Character codes will be mapped to integer values as follows: "X"->23, "XY"->24, "Y"-> 25, "M","MT"->26.
input.type	Format of input files. Accepted file types are "IMPUTE2", "BEAGLE", and "MaCH".
input.dosage	Logical for whether the genotype file (input.files[1]) contains dosages. If FALSE (default), the genotype file is assumed to contain genotype probabilities.
file.type	The type of file to create ("gds" or "ncdf")
snp.annot.filename	Output .RData file for storing a SnpAnnotationDataFrame.
scan.annot.filename	Output .RData file for storing a ScanAnnotationDataFrame.
precision	A character value indicating whether floating point numbers should be stored as "double" or "single" precision.
compress	compression method for GDS nodes, can be "", "ZIP", "ZIP.fast", "ZIP.default", "ZIP.max"
genotypeDim	character string specifying genotype dimensions of gds file. Either "snp,scan" or "scan,snp"
scan.df	data frame specifying which samples to include in the output GDS files, with optional scanIDs already assigned. See details.
snp.exclude	vector of integers specifying which SNPs to exclude from the GDS file.
snp.id.start	Starting index for snpID.
block.size	Number of lines to read at once.
verbose	Logical for whether to print progress messages.
genoData	A GenotypeData object from a GDS file created with imputedDosageFile.
snpAnnot	The SnpAnnotationDataFrame created by imputedDosageFile
scanAnnot	The ScanAnnotationDataFrame created by imputedDosageFile
tolerance	tolerance for checking differences against input files
na.logfile	filename for recording snpID and scanID of missing dosages

Details

Input files can contain either imputed dosages or genotype probabilities, specified by the `input.dosage` flag. In either case, the GDS/NetCDF file will store dosage of the A allele in the "genotype" variable. All SNPs are assumed to be on the same chromosome, which is indicated by the `chromosome` argument.

If the input file contains genotype probabilities for all three genotypes, the dosage is set to missing if the genotype probability strings (before numerical conversion) are equal (e.g., (0,0,0), (0.33, 0.33, 0.33), or (-1, -1, -1)). The dosage is also normalized by the sum of all three genotype probabilities.

The `scan.df` argument allows the user to specify what samples should be included in the GDS files and an optional sampleID-scanID mapping. `scan.df` is a data frame with required column `sampleID`. The function attempts to match the given sampleID in the `scan.df` data frame with a unique sampleID in the input files. The format of sampleID is different for different input types:

- IMPUTE2: "ID_1 ID_2" as given in the sample file, where IDs are separated by a space
- BEAGLE: Column header names corresponding to that sample in .dose or .gprobs file
- MaCH: The first column of the .mlprob or .mlprob file

The `snp.names` argument allows the user to specify the which SNPs should be included in the GDS files. However, `snp.names` must be in the same order as SNPs occur in the imputation files; this option therefore only allows selection of SNPs, not reordering of SNPs. The ordering is checked and an error is thrown if the SNP names are not in order, but due to the design of imputation files, this may not occur until well into the GDS file population. The user can specify the starting snpID by setting `snp.id.start`, and included SNPs are numbered sequentially starting with `snp.id.start`. For IMPUTE2 data, snp.names must correspond to the second column of the .gprobs file.

Minimal SNP and scan annotation are created from the input files and stored in RData format in `snp.annot.filename` and `scan.annot.filename`.

If requested with `na.logfile`, `checkImputedDosageFile` will output a file with scanIDs and snpIDs of missing genotype calls.

Currently supported input file types are IMPUTE2, BEAGLE, and MaCH.

Author(s)

Adrienne Stilp, Stephanie Gogarten

References

IMPUTE2: http://mathgen.stats.ox.ac.uk/impute/impute_v2.html

BEAGLE: http://faculty.washington.edu/browning/beagle/beagle.html

MaCH: http://www.sph.umich.edu/csg/abecasis/MACH/tour/imputation.html

See Also

`createDataFile`, `GdsGenotypeReader`, `NcdfGenotypeReader`, `GenotypeData`, `assocTestRegression`

Examples

```
gdsfile <- tempfile()
snpfile <- tempfile()
scanfile <- tempfile()
logfile <- tempfile()

# IMPUTE2
probfile <- system.file("extdata", "imputation", "IMPUTE2", "example.chr22.study.gens",
                        package="GWASdata")
sampfile <- system.file("extdata", "imputation", "IMPUTE2", "example.study.samples",
                        package="GWASdata")
imputedDosageFile(input.files=c(probfile, sampfile), filename=gdsfile, chromosome=22,
                  input.type="IMPUTE2", input.dosage=FALSE,
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)

gds <- GdsGenotypeReader(gdsfile)
scanAnnot <- getobj(scanfile)
snpAnnot <- getobj(snpfile)
genoData <- GenotypeData(gds, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

checkImputedDosageFile(genoData, snpAnnot, scanAnnot,
                       input.files=c(probfile, sampfile), chromosome=22,
                       input.type="IMPUTE2", input.dosage=FALSE, na.logfile=logfile)

geno <- getGenotype(genoData)
getVariable(genoData, "alleleA")
getVariable(genoData, "alleleB")

log <- read.table(logfile)
head(log)

# association test with imputed dosages
scanAnnot$status <- sample(0:1, nrow(scanAnnot), replace=TRUE)
genoData <- GenotypeData(gds, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
assoc <- assocTestRegression(genoData, outcome="status", model.type="logistic",
                             gene.action.list="additive", dosage=TRUE)
head(assoc)
close(genoData)


# BEAGLE - genotype probabilities
probfile <- system.file("extdata", "imputation", "BEAGLE", "example.hapmap.unphased.bgl.gprobs",
                        package="GWASdata")
markfile <- system.file("extdata", "imputation", "BEAGLE", "hapmap.markers",
                        package="GWASdata")
imputedDosageFile(input.files=c(probfile, markfile), filename=gdsfile, chromosome=22,
                  input.type="BEAGLE", input.dosage=FALSE, file.type="gds",
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)

# BEAGLE - dosage
dosefile <- system.file("extdata", "imputation", "BEAGLE", "example.hapmap.unphased.bgl.dose",
                        package="GWASdata")
```

```
imputedDosageFile(input.files=c(dosefile, markfile), filename=gdsfile, chromosome=22,
                  input.type="BEAGLE", input.dosage=TRUE, file.type="gds",
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)


# MaCH - genotype probabilities
probfile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mlprob",
                        package="GWASdata")
markfile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mlinfo",
                        package="GWASdata")
posfile <- system.file("extdata", "imputation", "MaCH", "mach1.snp.position",
                        package="GWASdata")
imputedDosageFile(input.files=c(probfile, markfile, posfile), filename=gdsfile, chromosome=22,
                  input.type="MaCH", input.dosage=FALSE, file.type="gds",
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)

# MaCH - dosage
dosefile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mldose",
                        package="GWASdata")
imputedDosageFile(input.files=c(dosefile, markfile, posfile), filename=gdsfile, chromosome=22,
                  input.type="MaCH", input.dosage=TRUE,  file.type="gds",
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)

unlink(c(gdsfile, snpfile, scanfile))
```

`IntensityData-class` *Class IntensityData*

Description

The IntensityData class is a container for storing intensity data from a genome-wide association study together with the metadata associated with the subjects and SNPs involved in the study.

Details

The IntensityData class consists of three slots: data, snp annotation, and scan annotation. There may be multiple scans associated with a subject (e.g. duplicate scans for quality control), hence the use of "scan" as one dimension of the data. Snp and scan annotation are optional, but if included in the IntensityData object, their unique integer ids (snpID and scanID) are checked against the ids stored in the data file to ensure consistency.

Constructor

`IntensityData(data, snpAnnot=NULL, scanAnnot=NULL)`:

`data` must be a `GdsIntensityReader` or `NcdfIntensityReader` object.

`snpAnnot`, if not `NULL`, must be a `SnpAnnotationDataFrame` or `SnpAnnotationSQLite` object.

`scanAnnot`, if not `NULL`, must be a `ScanAnnotationDataFrame` or `ScanAnnotationSQLite` object.

The IntensityData constructor creates and returns a IntensityData instance, ensuring that data, snpAnnot, and scanAnnot are internally consistent.

Accessors

In the code snippets below, object is an IntensityData object. snp and scan indicate which elements to return along the snp and scan dimensions. They must be integer vectors of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that the entire dimension should be read. If snp and/or is scan omitted, the entire variable is read.

nsnp(object): The number of SNPs in the data.

nscan(object): The number of scans in the data.

getSnpID(object, index): A unique integer vector of snp IDs. The optional index is a logical or integer vector specifying elements to extract.

getChromosome(object, index, char=FALSE): A vector of chromosomes. The optional index is a logical or integer vector specifying elements to extract. If char=FALSE (default), returns an integer vector. If char=TRUE, returns a character vector with elements in (1:22,X,XY,Y,M,U).

getPosition(object, index): An integer vector of base pair positions. The optional index is a logical or integer vector specifying elements to extract.

getScanID(object, index): A unique integer vector of scan IDs. The optional index is a logical or integer vector specifying elements to extract.

getSex(object, index): A character vector of sex, with values 'M' or 'F'. The optional index is a logical or integer vector specifying elements to extract.

hasSex(object): Returns TRUE if the column 'sex' is present in object.

getQuality(object, snp, scan): Extracts quality scores. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as NA.

getX(object, snp, scan): Extracts X intensity values. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as NA.

getY(object, snp, scan): Extracts Y intensity values. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as NA.

getBAlleleFreq(object, snp, scan): Extracts B allele frequency values. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as NA.

getLogRRatio(object, snp, scan): Extracts Log R Ratio values. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as NA.

getSnpVariable(object, varname, index): Returns the snp annotation variable varname. The optional index is a logical or integer vector specifying elements to extract.

getSnpVariableNames(object): Returns a character vector with the names of the columns in the snp annotation.

`hasSnpVariable(object, varname)`: Returns `TRUE` if the variable `varname` is present in the snp annotation.

`getScanVariable(object, varname, index)`: Returns the scan annotation variable `varname`. The optional `index` is a logical or integer vector specifying elements to extract.

`getScanVariableNames(object)`: Returns a character vector with the names of the columns in the scan annotation.

`hasScanVariable(object, varname)`: Returns `TRUE` if the variable `varname` is present in the scan annotation.

`getVariable(object, varname, snp, scan)`: Extracts the contents of the variable `varname` from the data. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`. If the variable is not found, returns `NULL`.

`hasVariable(object, varname)`: Returns `TRUE` if the data contains contains `varname`, `FALSE` if not.

`hasSnpAnnotation(object)`: Returns `TRUE` if the snp annotation slot is not `NULL`.

`hasScanAnnotation(object)`: Returns `TRUE` if the scan annotation slot is not `NULL`.

`open(object)`: Opens a connection to the data.

`close(object)`: Closes the data connection.

`autosomeCode(object)`: Returns the integer codes for the autosomes.

`XchromCode(object)`: Returns the integer code for the X chromosome.

`XYchromCode(object)`: Returns the integer code for the pseudoautosomal region.

`YchromCode(object)`: Returns the integer code for the Y chromosome.

`MchromCode(object)`: Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

`SnpAnnotationDataFrame`, `SnpAnnotationSQLite`, `ScanAnnotationDataFrame`, `ScanAnnotationSQLite`, `ScanAnnotationDataFrame`, `GdsIntensityReader`, `NcdfIntensityReader`, `GenotypeData`

Examples

```
library(GWASdata)
file <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
gds <- GdsIntensityReader(file)

# object without annotation
intenData <- IntensityData(gds)

# object with annotation
data(illuminaSnpADF, illuminaScanADF)
intenData <-  IntensityData(gds, snpAnnot=illuminaSnpADF, scanAnnot=illuminaScanADF)
```

```
# dimensions
nsnp(intenData)
nscan(intenData)

# get snpID and chromosome
snpID <- getSnpID(intenData)
chrom <- getChromosome(intenData)

# get positions only for chromosome 22
pos22 <- getPosition(intenData, index=(chrom == 22))

# get other annotations
if (hasSex(intenData)) sex <- getSex(intenData)
plate <- getScanVariable(intenData, "plate")
rsID <- getSnpVariable(intenData, "rsID")

# get all snps for first scan
x <- getX(intenData, snp=c(1,-1), scan=c(1,1))

# starting at snp 100, get 10 snps for the first 5 scans
x <- getX(intenData, snp=c(100,10), scan=c(1,5))

close(intenData)
```

`intensityOutliersPlot` *Plot mean intensity and highlight outliers*

Description

`intensityOutliersPlot` is a function to plot mean intensity for chromosome i vs mean of intensities for autosomes (excluding i) and highlight outliers

Usage

```
intensityOutliersPlot(mean.intensities, sex, outliers,
              sep = FALSE, label, ...)
```

Arguments

`mean.intensities`	scan x chromosome matrix of mean intensities
`sex`	vector with values of "M" or "F" corresponding to scans in the rows of `mean.intensities`
`outliers`	list of outliers, each member corresponds to a chromosome (member "X" is itself a list of female and male outliers)
`sep`	plot outliers within a chromosome separately (TRUE) or together (FALSE)
`label`	list of plot labels (to be positioned below X axis) corresponding to list of outliers
`...`	additional arguments to `plot`

Details

Outliers must be determined in advance and stored as a list, with one element per chromosome. The X chromosome must be a list of two elements, "M" and "F". Each element should contain a vector of ids corresponding to the row names of `mean.intensities`.

If `sep=TRUE`, `labels` must also be specified. `labels` should be a list that corresponds exactly to the elements of `outliers`.

Author(s)

Cathy Laurie

See Also

`meanIntensityByScanChrom`

Examples

```
# calculate mean intensity
library(GWASdata)
file <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
gds <- GdsIntensityReader(file)
data(illuminaScanADF)
intenData <- IntensityData(gds, scanAnnot=illuminaScanADF)
meanInten <- meanIntensityByScanChrom(intenData)
intenMatrix <- meanInten$mean.intensity

# find outliers
outliers <- list()
sex <- illuminaScanADF$sex
id <- illuminaScanADF$scanID
allequal(id, rownames(intenMatrix))
for (i in colnames(intenMatrix)) {
  if (i != "X") {
    imean <- intenMatrix[,i]
    imin <- id[imean == min(imean)]
    imax <- id[imean == max(imean)]
    outliers[[i]] <- c(imin, imax)
  } else {
    idf <- id[sex == "F"]
    fmean <- intenMatrix[sex == "F", i]
    fmin <- idf[fmean == min(fmean)]
    fmax <- idf[fmean == max(fmean)]
    outliers[[i]][["F"]] <- c(fmin, fmax)
    idm <- id[sex == "M"]
    mmean <- intenMatrix[sex == "M", i]
    mmin <- idm[mmean == min(mmean)]
    mmax <- idm[mmean == max(mmean)]
    outliers[[i]][["M"]] <- c(mmin, mmax)
  }
}
```

```
par(mfrow=c(2,4))
intensityOutliersPlot(intenMatrix, sex, outliers)
close(intenData)
```

`manhattanPlot` *Manhattan plot for genome wide association tests*

Description

Generates a manhattan plot of the results of a genome wide association test.

Usage

```
manhattanPlot(p, chromosome,
              ylim = NULL, trunc.lines = TRUE,
              signif = 5e-8, thinThreshold=NULL, pointsPerBin=10000, ...)
```

Arguments

`p`	A vector of p-values.
`chromosome`	A vector containing the chromosome for each SNP.
`ylim`	The limits of the y axis. If NULL, the y axis is `(0, log10(length(p)) + 4)`.
`trunc.lines`	Logical value indicating whether to show truncation lines.
`signif`	Genome-wide significance level for plotting horizontal line. If `signif=NULL`, no line will be drawn.
`thinThreshold`	if not `NULL`, `-log10(pval)` threshold for thinning points.
`pointsPerBin`	number of points to plot in each bin if `thinThreshold` is given. Ignored otherwise.
`...`	Other parameters to be passed directly to `plot`.

Details

Plots -log10(p) versus chromosome. Point size is scaled so that smaller p values have larger points.

`p` must have the same length as `chromosome` and is assumed to be in order of position on each chromosome. Values within each chromosome are evenly spaced along the X axis.

Plot limits are determined as follows: if `ylim` is provided, any points with `-log10(p) > ylim[2]` are plotted as triangles at the maximum y value of the plot. A line will be drawn to indicate truncation (if `trunc.lines == TRUE`, the default). If `ylim == NULL`, the maximum y value is defined as `log10(length(p)) + 4`).

If requested with `thinThreshold`, points with `-log10(pval) < thinThreshold` are thinned before plotting. All points with `-log10(pval) >= thinThreshold` are displayed. P-values with `-log10(pval) < thinThreshold` are sampled such that `pointsPerBin` points are randomly selected from 10 bins with uniform spacing in `-log10(pval)` space.

Author(s)

Cathy Laurie, Adrienne Stilp

See Also

snpCorrelationPlot

Examples

```
n <- 1000
pvals <- sample(-log10((1:n)/n), n, replace=TRUE)
chromosome <- c(rep(1,400), rep(2,350), rep("X",200), rep("Y",50))
manhattanPlot(pvals, chromosome, signif=1e-7)
manhattanPlot(pvals, chromosome, thinThreshold=2)
```

MatrixGenotypeReader *Class MatrixGenotypeReader*

Description

The MatrixGenotypeReader class stores a matrix of genotypes as well as SNP and scan IDs, chromosome, and position.

Constructor

`MatrixGenotypeReader(genotype=genotype, snpID=snpID, chromosome=chromosome, position=position, scanI`

`genotype` must be a matrix with dimensions ('snp','scan') containing the number of A alleles : 2=AA, 1=AB, 0=BB.

`snp` must be a unique integer vector of SNP ids.

`chromosome` must be an integer vector of chromosomes. Default values for chromosome codes are 1-22=autosome, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `autosomeCode`, `XchromCode`, `XYchromCode`, `YchromCode`, and `MchromCode`.

`position` must be an integer vector of base positions

`scanID` must be a unique integer vector of scan ids .

The `MatrixGenotypeReader` constructor creates and returns a MatrixGenotypeReader instance.

Accessors

In the code snippets below, `object` is a MatrixGenotypeReader object. `snp` and `scan` indicate which elements to return along the snp and scan dimensions. They must be integer vectors of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that the entire dimension should be read. If snp and/or is scan omitted, the entire variable is returned.

See NcdfReader for additional methods.

`nsnp(object)`: The number of SNPs.

`nscan(object)`: The number of scans.

`getSnpID(object, index)`: A unique integer vector of snp IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getChromosome(object, index, char=FALSE)`: A vector of chromosomes. The optional `index` is a logical or integer vector specifying elements to extract. If `char=FALSE` (default), returns an integer vector. If `char=TRUE`, returns a character vector with elements in (1:22,X,XY,Y,M,U). "U" stands for "Unknown" and is the value given to any chromosome code not falling in the other categories.

`getPosition(object, index)`: An integer vector of base pair positions. The optional `index` is a logical or integer vector specifying elements to extract.

`getScanID(object, index)`: A unique integer vector of scan IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getGenotype(object, snp, scan)`: Extracts genotype values (number of A alleles). The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

`autosomeCode(object)`: Returns the integer codes for the autosomes.

`XchromCode(object)`: Returns the integer code for the X chromosome.

`XYchromCode(object)`: Returns the integer code for the pseudoautosomal region.

`YchromCode(object)`: Returns the integer code for the Y chromosome.

`McchromCode(object)`: Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

`NcdfGenotypeReader`, `GenotypeData`

Examples

```
snpID <- 1:100
chrom <- rep(1:20, each=5)
pos <- 1001:1100
scanID <- 1:20
geno <- matrix(sample(c(0,1,2,NA), 2000, replace=TRUE), nrow=100, ncol=20)

mgr <- MatrixGenotypeReader(genotype=geno, snpID=snpID,
  chromosome=chrom, position=pos, scanID=scanID)

# dimensions
nsnp(mgr)
nscan(mgr)

# get snpID and chromosome
snpID <- getSnpID(mgr)
chrom <- getChromosome(mgr)
```

```
# get positions only for chromosome 10
pos10 <- getPosition(mgr, index=(chrom == 10))

# get all snps for first scan
geno <- getGenotype(mgr, snp=c(1,-1), scan=c(1,1))

# starting at snp 50, get 10 snps for the first 5 scans
geno <- getGenotype(mgr, snp=c(50,10), scan=c(1,5))
```

`meanIntensityByScanChrom` *Calculate Means \& Standard Deviations of Intensities*

Description

Function to calculate the mean and standard deviation of the intensity for each chromosome for each scan.

Usage

```
meanIntensityByScanChrom(intenData, vars = c("X", "Y"),
                         snp.exclude = NULL, verbose = TRUE)
```

Arguments

`intenData`	`IntensityData` object. Chromosomes are expected to be in contiguous blocks.
`vars`	Character vector with the names of one or two intensity variables.
`snp.exclude`	An integer vector containing SNPs to be excluded.
`verbose`	Logical value specifying whether to show progress information.

Details

The names of two intensity variables in `intenData` may be supplied. If two variables are given, the mean of their sum is computed as well. The default is to compute the mean and standard deviation for X and Y intensity.

Value

A list with two components for each variable in "vars": 'mean.var' and 'sd.var'. If two variables are given, the first two elements of the list will be mean and sd for the sum of the intensity variables:

`mean.intensity`	A matrix with one row per scan and one column per chromosome containing the means of the summed intensity values for each scan and chromosome.
`sd.intensity`	A matrix with one row per scan and one column per chromosome containing the standard deviations of the summed intensity values for each scan and chromosome.

mean.var	A matrix with one row per scan and one column per chromosome containing the means of the intensity values for each scan and chromosome.
sd.var	A matrix with one row per scan and one column per chromosome containing the standard deviations of the intensity values for each scan and chromosome.

Author(s)

Cathy Laurie

See Also

IntensityData, mean, sd

Examples

```
file <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
gds <- GdsIntensityReader(file)
intenData <- IntensityData(gds)

meanInten <- meanIntensityByScanChrom(intenData)
close(intenData)
```

---

mendelErr *Mendelian Error Checking*

Description

Mendelian and mtDNA inheritance tests.

Usage

```
mendelErr(genoData, mendel.list, snp.exclude=NULL,
          error.by.snp=TRUE, error.by.snp.trio=FALSE,
          verbose=TRUE, outfile=NULL)
```

Arguments

genoData	GenotypeData object, must have scan variable "sex"
mendel.list	A mendelList object, to specify trios.
snp.exclude	An integer vector with snpIDs of SNPs to exclude. If NULL (default), all SNPs are used.
error.by.snp	Whether or not to output Mendelian errors per SNP. This will only return the total number of trios checked and the total number of errors for each SNP. The default value is TRUE.

`error.by.snp.trio`
: Whether or not to output Mendelian errors per SNP for each trio. This will return the total number of trios checked and the total number of errors for each SNP as well as indicators of which SNPs have an error for each trio. The default value is `FALSE`. NOTE: `error.by.snp` must be set to `TRUE` as well in order to use this option. NOTE: Using this option causes the output to be very large that may be slow to load into R.

`verbose`
: If `TRUE` (default), will print status updates while the function runs.

`outfile`
: A character string to append in front of ".RData" for naming the output file.

Details

`genoData` must contain the scan annotation variable "sex". Chromosome index: 1..22 autosomes, 23 X, 24 XY, 25 Y, 26 mtDNA, 27 missing.

Another file will be saved with the name "outfile.warnings.RData" that contains any warnings generated by the function.

Value

If `outfile=NULL` (default), `mendelErr` returns an object of class "mendelClass". If `outfile` is specified, no data is returned but `mendelErr` saves the object to disk as "outfile.RData."

The object contains two data frames: "trios" and "all.trios", and a list: "snp" (if `error.by.snp` is specified to be `TRUE`). If there are no duplicate samples in the dataset, "trios" will be the same as "all.trios". Otherwise, "all.trios" contains the results of all combinations of duplicate samples, and "trios" only stores the average values of unique trios. i.e: "trios" averages duplicate samples for each unique subject trio. "trios" and "all.trios" contain the following components:

`fam.id`
: Specifying the family ID from the mendel.list object used as input.

`child.id`
: Specifying the offspring ID from the mendel.list object used as input.

`child.scanID`
: Specifying the offspring scanID from the mendel.list object used as input. (only in "all.trios")

`father.scanID`
: Specifying the father scanID from the mendel.list object used as input. (only in "all.trios")

`mother.scanID`
: Specifying the mother scanID from the mendel.list object used as input. (only in "all.trios")

`Men.err.cnt`
: The number of SNPs with Mendelian errors in this trio.

`Men.cnt`
: The total number of SNPs checked for Mendelian errors in this trio. It excludes those cases where the SNP is missing in the offspring and those cases where it is missing in both parents. Hence, Mendelian error rate = `Men.err.cnt` / `Men.cnt`.

`mtDNA.err`
: The number of SNPs with mtDNA inheritance errors in this trio.

`mtDNA.cnt`
: The total number of SNPs checked for mtDNA inheritance errors in this trio. It excludes those cases where the SNP is missing in the offspring and in the mother. Hence, mtDNA error rate = `mtDNA.err` / `mtDNA.cnt` .

`chr1, ..., chr25`
: The number of Mendelian errors in each chromosome for this trio.

"snp" is a list that contains the following components:

`check.cnt` A vector of integers, indicating the number of trios valid for checking on each SNP.

`error.cnt` A vector of integers, indicating the number of trios with errors on each SNP.

`familyid.childid`
A vector of indicators (0/1) for whether or not any of the duplicate trios for the unique trio, "familyid.childid", have a Mendelian error on each SNP. (Only if `error.by.snp.trio` is specified to be `TRUE`).

Warnings:

If `outfile` is not `NULL`, another file will be saved with the name "outfile.warnings.RData" that contains any warnings generated by the function.

Author(s)

Xiuwen Zheng, Matthew P. Conomos

See Also

`mendelList`

Examples

```
library(GWASdata)
data(illuminaScanADF)
scanAnnot <- illuminaScanADF

# generate trio list
men.list <- mendelList(scanAnnot$family, scanAnnot$subjectID,
  scanAnnot$father, scanAnnot$mother, scanAnnot$sex,
  scanAnnot$scanID)

# create genoData object
gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(gdsfile)
genoData <- GenotypeData(gds, scanAnnot=scanAnnot)

# Run!
outfile <- tempfile()
mendelErr(genoData, men.list, error.by.snp.trio = TRUE, outfile =
outfile)

# Load the output
R <- getobj(paste(outfile, "RData", sep="."))
names(R)
# [1] "trios"     "all.trios" "snp"

names(R$trios)
#  [1] "fam.id"      "child.id"    "Men.err.cnt" "Men.cnt"     "mtDNA.err"
#  [6] "mtDNA.cnt"   "chr1"        "chr2"        "chr3"        "chr4"
```

```
# [11] "chr5"        "chr6"        "chr7"        "chr8"        "chr9"
# [16] "chr10"       "chr11"       "chr12"       "chr13"       "chr14"
# [21] "chr15"       "chr16"       "chr17"       "chr18"       "chr19"
# [26] "chr20"       "chr21"       "chr22"       "chr23"       "chr24"
# [31] "chr25"

# Mendelian error rate = Men.err.cnt / Men.cnt
data.frame(fam.id = R$trios$fam.id, child.id = R$trios$child.id,
           Mendel.err.rate = R$trios$Men.err.cnt / R$trios$Men.cnt)

names(R$snp)
summary(R$snp$check.cnt)

# summary Mendelian error for first family
summary(R$snp[[1]])

# check warnings
warnfile <- paste(outfile, "warnings.RData", sep=".")
if (file.exists(warnfile)) warns <- getobj(warnfile)

close(genoData)
unlink(paste(outfile, "*", sep=""))
```

`mendelList` *Mendelian Error Checking*

Description

`mendelList` creates a "mendelList" object (a list of trios). `mendelListAsDataFrame` converts a "mendelList" object to a data frame.

Usage

```
mendelList(familyid, offspring, father, mother, sex, scanID)

mendelListAsDataFrame(mendel.list)
```

Arguments

`familyid`	A vector of family identifiers.
`offspring`	A vector of offspring subject identifiers.
`father`	A vector of father identifiers.
`mother`	A vector of mother identifiers.
`sex`	A vector to specify whether each subject is male "M" or female "F".
`scanID`	A vector of scanIDs indicating unique genotyping instances for the `offspring` vector. In the case of duplicate samples, the same `offspring` identifier may correspond to multiple `scanID` values.
`mendel.list`	An object of class "mendelList".

Details

The lengths of `familyid`, `offspring`, `father`, `mother`, `sex`, and `scanID` must all be identical. These vectors should include all genotyped samples, i.e., samples present in the `father` and `mother` vectors should also appear in the `offspring` vector if there are genotypes for these samples, and their unique scan IDs should be given in the `scanID` vector.

Identifiers may be character strings or integers, but not factors.

The "mendelList" object is required as input for the `mendelErr` function.

Value

`mendelList` returns a "mendelList" object. A "mendelList" object is a list of lists. The first level list is all the families. The second level list is offspring within families who have one or both parents genotyped. Within the second level are data.frame(s) with columns "offspring", "father", and "mother" which each contain the scanID for each member of the trio (a missing parent is denoted by -1). When replicates of the same offsping ID occur (duplicate scans for the same subject), this data.frame has multiple rows representing all combinations of scanIDs for that trio.

`mendelListAsDataFrame` returns a data.frame with variables "offspring", "father", and "mother" which each contain the scanID for each member of the trio (a missing parent is denoted by -1). This takes every data.frame from the "mendelList" object and puts them all into one large data frame. This can be easier to work with for certain analyses.

Author(s)

Xiuwen Zheng, Matthew P. Conomos

See Also

`mendelErr`

Examples

```
# data frame of sample information.  No factors!
dat <- data.frame(family=c(1,1,1,1,2,2,2), offspring=c("a","a","b","c","d","e","f"),
  father=c("b","b",0,0,"e",0,0), mother=c("c","c",0,0,"f",0,0),
  sex=c("M","M","M","F","F","M","F"), scanID=1:7,
  stringsAsFactors=FALSE)
dat

men.list <- mendelList(dat$family, dat$offspring, dat$father, dat$mother,
                       dat$sex, dat$scanID)
men.list

# If fathers and mothers do not have separate entries in each vector,
# mendelList returns a "NULL":
dat <- dat[c(1,5),]
dat
mendelList(dat$family, dat$offspring, dat$father, dat$mother,
           dat$sex, dat$scanID)
```

```
men.df <- mendelListAsDataFrame(men.list)
men.df
```

`missingGenotypeByScanChrom` *Missing Counts per Scan per Chromosome*

Description

This function tabulates missing genotype calls for each scan for each chromosome.

Usage

```
missingGenotypeByScanChrom(genoData, snp.exclude = NULL,
                           verbose = TRUE)
```

Arguments

`genoData`	`GenotypeData` object. Chromosomes are expected to be in contiguous blocks.
`snp.exclude`	A vector of IDs corresponding to the SNPs that should be excluded from the overall missing count.
`verbose`	Logical value specifying whether to show progress information.

Details

This function calculates the percent of missing genotypes in each chromosome of each scan given in `genoData`. A "sex" variable must be present in the scan annotation slot of `genoData`.

Value

This function returns a list with three components: "missing.counts," "snps.per.chr", and "missing.fraction."

`missing.counts`	A matrix with rows corresponding to the scans and columns indicating unique chromosomes containing the number of missing SNP's for each scan and chromosome.
`snps.per.chr`	A vector containing the number of non-excluded SNPs for each chromosome.
`missing.fraction`	A vector containing the fraction of missing counts for each scan over all chromosomes, excluding the Y chromosome for females.

Author(s)

Cathy Laurie

See Also

`GenotypeData`, `missingGenotypeBySnpSex`

Examples

```
library(GWASdata)
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)

# need scan annotation with sex
data(illuminaScanADF)
genoData <-  GenotypeData(gds, scanAnnot=illuminaScanADF)

missingRate <- missingGenotypeByScanChrom(genoData)
close(genoData)
```

---

missingGenotypeBySnpSex

Missing Counts per SNP by Sex

Description

For all SNPs for each sex tabulates missing SNP counts, allele counts and heterozygous counts.

Usage

```
missingGenotypeBySnpSex(genoData, scan.exclude = NULL,
                        verbose = TRUE)
```

Arguments

genoData	GenotypeData object.
scan.exclude	A vector containing the scan numbers of scans that are to be excluded from the total scan list.
verbose	Logical value specifying whether to show progress information.

Details

This function calculates the fraction of missing genotypes for males and females for each SNP given in genoData. A "sex" variable must be present in the scan annotation slot of genoData.

Value

This function returns a list with three components: "missing.counts," "scans.per.sex," and "missing.fraction."

missing.counts	A matrix with one row per SNP and one column per sex containing the number of missing SNP counts for males and females, respectively.
scans.per.sex	A vector containing the number of males and females respectively.
missing.fraction	A vector containing the fraction of missing counts for each SNP, with females excluded for the Y chromosome.

Author(s)

Cathy Laurie, Stephanie Gogarten

See Also

GenotypeData, missingGenotypeByScanChrom

Examples

```
library(GWASdata)
file <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(file)

# need scan annotation with sex
data(illuminaScanADF)
genoData <-  GenotypeData(gds, scanAnnot=illuminaScanADF)

missingRate <- missingGenotypeBySnpSex(genoData)
close(genoData)
```

---

ncdfAddData — *Write genotypic calls and/or associated metrics to a netCDF file - deprecated*

Description

Deprecated - use createDataFile, createAffyIntensityFile, checkGenotypeFile, checkIntensityFile.

Genotypic calls and/or associated quantitative variables (e.g. quality score, intensities) are read from text files and written to an existing netCDF file in which those variables were defined previously.

Usage

```
ncdfAddData(path = ".", ncdf.filename,
            snp.annotation, scan.annotation,
            sep.type, skip.num, col.total, col.nums,
            scan.name.in.file, scan.start.index = 1,
            diagnostics.filename = "ncdfAddData.diagnostics.RData",
            verbose = TRUE)

ncdfAddIntensity(path = ".",  ncdf.filename,
                 snp.annotation, scan.annotation,
                 scan.start.index = 1, n.consecutive.scans = -1,
                 diagnostics.filename = "ncdfAddIntensity.diagnostics.RData",
                 verbose = TRUE)

ncdfCheckGenotype(path = ".", ncdf.filename,
                  snp.annotation, scan.annotation,
```

```
                    sep.type, skip.num, col.total, col.nums,
                    scan.name.in.file, check.scan.index, n.scans.loaded,
                    diagnostics.filename = "ncdfCheckGenotype.diagnostics.RData",
                    verbose = TRUE)

ncdfCheckIntensity(path = ".", intenpath = ".", ncdf.filename,
                     snp.annotation, scan.annotation,
                     sep.type, skip.num, col.total, col.nums,
                     scan.name.in.file, check.scan.index,
                     n.scans.loaded, affy.inten = FALSE,
                   diagnostics.filename = "ncdfCheckIntensity.diagnostics.RData",
                     verbose = TRUE)
```

Arguments

path	Path to the raw text files.
intenpath	Path to the raw text files containing intensity, if "inten.file" is given in scan.annotation.
ncdf.filename	Name of the netCDF file in which to write the data.
snp.annotation	SNP annotation data.frame containing SNPs in the same order as those in the snp dimension of the netCDF file. Column names must be "snpID" (integer ID) and "snpName", where snpName matches the snp ids inside the raw genoypic data files.
scan.annotation	Scan annotation data.frame with columns "scanID" (integer id of scan in the netCDF file), "scanName", (sample name inside the raw data file) and "file" (corresponding raw data file name).
sep.type	Field separator in the raw text files.
skip.num	Number of rows to skip, which should be all rows preceding the genotypic or quantitative data (including the header).
col.total	Total number of columns in the raw text files.
col.nums	An integer vector indicating which columns of the raw text file contain variables for input. `names(col.nums)` must be a subset of c("snp", "sample", "geno", "a1", "a2", "qs", "x", "y", "rawx", "rawy", "r", "theta", "ballelefreq", "logrratio"). The element "snp" is the column of SNP ids, "sample" is sample ids, "geno" is diploid genotype (in AB format), "a1" and "a2" are alleles 1 and 2 (in AB format), "qs" is quality score, "x" and "y" are normalized intensities, "rawx" and "rawy" are raw intensities, "r" is the sum of normalized intensities, "theta" is angular polar coordinate, "ballelefreq" is the B allele frequency, and "logrratio" is the Log R Ratio.
scan.name.in.file	An indicator for the presence of sample name within the file. A value of 1 indicates a column with repeated values of the sample name (Illumina format), -1 indicates sample name embedded in a column heading (Affymetrix format) and 0 indicates no sample name inside the raw data file.
scan.start.index	A numeric value containing the index of the sample dimension of the netCDF file at which to begin writing.

`n.consecutive.scans`
: The number of consecutive "sampleID" indices for which to write intensity values, beginning at scan.start.index (which equals the number of "ALLELE_SUMMARY" files to process). When n.consecutive.scans=-1, all samples from scan.start.index to the total number will be processed.

`check.scan.index`
: An integer vector containing the indices of the sample dimension of the netCDF file to check.

`n.scans.loaded`
: Number of scans loaded in the netCDF file.

`affy.inten`
: Logical value indicating whether Affy intensities are in separate files from quality scores. If `TRUE`, must also specify intenpath.

`diagnostics.filename`
: Name of the output file to save diagnostics.

`verbose`
: Logical value specifying whether to show progress information.

Details

These functions read genotypic and associated data from raw text files. The files to be read and processed are specified in the sample annotation. `ncdfAddData` expects one file per sample, with each file having one row of data per SNP probe. The `col.nums` argument allows the user to select and identify specific fields for writing to the netCDF file. Illumina text files and Affymetrix ".CHP" files can be used here (but not Affymetrix "ALLELE_SUMMARY" files).

A SNP annotation data.frame is a pre-requisite for this function. It has the same number of rows (one per SNP) as the raw text file and a column of SNP names matching those within the raw text file. It also has a column of integer SNP ids matching the values (in order) of the "snp" dimension of the netCDF file.

A sample annotation data.frame is also a pre-requisite. It has one row per sample with columns corresponding to sample name (as it occurs within the raw text file), name of the raw text file for that sample and an integer sample id (to be written as the "sampleID" variable in the netCDF file).

The genotype calls in the raw text file may be either one column of diploid calls or two columns of allele calls. The function takes calls in AB format and converts them to a numeric code indicating the number of "A" alleles in the genotype (i.e. AA=2, AB=1, BB=0 and missing=-1).

While each raw text file is being read, the functions check for errors and irregularities and records the results in a list of vectors. If any problem is detected, that raw text file is skipped.

`ncdfAddIntensity` uses `scan.start.index` and `n.consecutive.scans` to identify the set of integer sample ids for input (from the netCDF file). It then uses the sample annotation data.frame to identify the corresponding sample names and "ALLELE_SUMMARY" file names to read. The "ALLELE_SUMMARY" files have two rows per SNP, one for X (A allele) and one for Y (B allele). These are reformatted to one row per SNP and and ordered according to the SNP integer id in the netCDF file. The correspondence between SNP names in the "ALLELE_SUMMARY" file and the SNP integer ids is made using the SNP annotation data.frame.

`ncdfCheckGenotype` and `ncdfCheckIntensity` check the contents of netCDF files against raw text files.

These functions use the **ncdf** library, which provides an interface between R and netCDF.

Value

The netCDF file specified in argument `ncdf.filename` is populated with genotype calls and/or associated quantitative variables. A list of diagnostics with the following components is returned. Each vector has one element per raw text file processed.

`read.file`	A vector indicating whether (1) or not (0) each file was read successfully.
`row.num`	A vector of the number of rows read from each file. These should all be the same and equal to the number of rows in the SNP annotation data.frame.
`samples`	A list of vectors containing the unique sample names in the sample column of each raw text file. Each vector should have just one element.
`sample.match`	A vector indicating whether (1) or not (0) the sample name inside the raw text file matches that in the sample annotation data.frame
`missg`	A list of vectors containing the unique character string(s) for missing genotypes (i.e. not AA,AB or BB) for each raw text file.
`snp.chk`	A vector indicating whether (1) or not (0) the raw text file has the expected set of SNP names (i.e. matching those in the SNP annotation data.frame).
`chk`	A vector indicating whether (1) or not (0) all previous checks were successful and the data were written to the netCDF file.

`ncdfCheckGenotypes` returns the following additional list items.

`snp.order`	A vector indicating whether (1) or not (0) the snp ids are in the same order in each file.
`geno.chk`	A vector indicating whether (1) or not (0) the genotypes in the netCDF match the text file.

`ncdfCheckIntensity` returns the following additional list items.

`qs.chk`	A vector indicating whether (1) or not (0) the quality scores in the netCDF match the text file.
`read.file.inten`	A vector indicating whether (1) or not (0) each intensity file was read successfully (if intensity files are separate).
`sample.match.inten`	A vector indicating whether (1) or not (0) the sample name inside the raw text file matches that in the sample annotation data.frame (if intensity files are separate).
`rows.equal`	A vector indicating whether (1) or not (0) the number of rows read from each file are the same and equal to the number of rows in the SNP annotation data.frame (if intensity files are separate).
`snp.chk.inten`	A vector indicating whether (1) or not (0) the raw text file has the expected set of SNP names (i.e. matching those in the SNP annotation data.frame) (if intensity files are separate).
`inten.chk`	A vector for each intensity variable indicating whether (1) or not (0) the intensities in the netCDF match the text file.

Note

These functions were modeled after similar code written by Thomas Lumley.

Author(s)

Cathy Laurie

See Also

ncdf, ncdfCreate, ncdfSubset

Examples

```
## Not run:
library(GWASdata)

#############
# Illumina - genotype file
#############
# first create empty netCDF
data(illumina_snp_annot)
snpAnnot <- illumina_snp_annot
data(illumina_scan_annot)
scanAnnot <- illumina_scan_annot[1:3,] # subset of samples for testing
ncfile <- tempfile()
ncdfCreate(snpAnnot, ncfile, variables="genotype",
               n.samples=nrow(scanAnnot))

# add data
path <- system.file("extdata", "illumina_raw_data", package="GWASdata")
snpAnnot <- snpAnnot[,c("snpID", "rsID")]
names(snpAnnot) <- c("snpID", "snpName")
scanAnnot <- scanAnnot[,c("scanID", "genoRunID", "file")]
names(scanAnnot) <- c("scanID", "scanName", "file")
col.nums <- as.integer(c(1,2,12,13))
names(col.nums) <- c("snp", "sample", "a1", "a2")
diagfile <- tempfile()
res <- ncdfAddData(path, ncfile, snpAnnot, scanAnnot, sep.type=",",
                     skip.num=11, col.total=21, col.nums=col.nums,
                     scan.name.in.file=1, diagnostics.filename=diagfile)

file.remove(diagfile)
file.remove(ncfile)

#############
# Affymetrix - genotype file
#############
# first create empty netCDF
data(affy_snp_annot)
snpAnnot <- affy_snp_annot
data(affy_scan_annot)
scanAnnot <- affy_scan_annot[1:3,] # subset of samples for testing
```

```
ncfile <- tempfile()
ncdfCreate(snpAnnot, ncfile, variables="genotype",
               n.samples=nrow(scanAnnot))

# add data
path <- system.file("extdata", "affy_raw_data", package="GWASdata")
snpAnnot <- snpAnnot[,c("snpID", "probeID")]
names(snpAnnot) <- c("snpID", "snpName")
scanAnnot <- scanAnnot[,c("scanID", "genoRunID", "chpFile")]
names(scanAnnot) <- c("scanID", "scanName", "file")
col.nums <- as.integer(c(2,3)); names(col.nums) <- c("snp", "geno")
diagfile <- tempfile()
res <- ncdfAddData(path, ncfile, snpAnnot, scanAnnot, sep.type="\t",
                    skip.num=1, col.total=6, col.nums=col.nums,
                    scan.name.in.file=-1, diagnostics.filename=diagfile)
file.remove(diagfile)

# check
diagfile <- tempfile()
res <- ncdfCheckGenotype(path, ncfile, snpAnnot, scanAnnot, sep.type="\t",
                      skip.num=1, col.total=6, col.nums=col.nums,
                      scan.name.in.file=-1, check.scan.index=1:3,
                      n.scans.loaded=3, diagnostics.filename=diagfile)
file.remove(diagfile)
file.remove(ncfile)

#############
# Affymetrix - intensity file
#############
# first create empty netCDF
snpAnnot <- affy_snp_annot
scanAnnot <- affy_scan_annot[1:3,] # subset of samples for testing
ncfile <- tempfile()
ncdfCreate(snpAnnot, ncfile, variables=c("quality","X","Y"),
               n.samples=nrow(scanAnnot))

# add sampleID and quality
path <- system.file("extdata", "affy_raw_data", package="GWASdata")
snpAnnot <- snpAnnot[,c("snpID", "probeID")]
names(snpAnnot) <- c("snpID", "snpName")
scanAnnot1 <- scanAnnot[,c("scanID", "genoRunID", "chpFile")]
names(scanAnnot1) <- c("scanID", "scanName", "file")
col.nums <- as.integer(c(2,4)); names(col.nums) <- c("snp", "qs")
diagfile <- tempfile()
res <- ncdfAddData(path, ncfile, snpAnnot, scanAnnot1, sep.type="\t",
                    skip.num=1, col.total=6, col.nums=col.nums,
                    scan.name.in.file=-1, diagnostics.filename=diagfile)
file.remove(diagfile)

# add intensity
scanAnnot2 <- scanAnnot[,c("scanID", "genoRunID", "alleleFile")]
names(scanAnnot2) <- c("scanID", "scanName", "file")
diagfile <- tempfile()
```

```
res <- ncdfAddIntensity(path, ncfile, snpAnnot, scanAnnot2,
                        diagnostics.filename=diagfile)
file.remove(diagfile)

# check
intenpath <- system.file("extdata", "affy_raw_data", package="GWASdata")
scanAnnot <- scanAnnot[,c("scanID", "genoRunID", "chpFile", "alleleFile")]
names(scanAnnot) <- c("scanID", "scanName", "file", "inten.file")
diagfile <- tempfile()
res <- ncdfCheckIntensity(path, intenpath, ncfile, snpAnnot, scanAnnot, sep.type="\t",
                        skip.num=1, col.total=6, col.nums=col.nums,
                        scan.name.in.file=-1, check.scan.index=1:3,
                        n.scans.loaded=3, affy.inten=TRUE,
                        diagnostics.filename=diagfile)

file.remove(diagfile)
file.remove(ncfile)

## End(Not run)
```

`ncdfCreate` *Write genotypic calls and/or associated metrics to a netCDF file - deprecated*

Description

Deprecated - use `createDataFile`

The function creates a shell netCDF file to which data can subsequently written.

Usage

```
ncdfCreate(snp.annotation, ncdf.filename, variables = "genotype",
           n.samples = 10, precision = "double",
           array.name = NULL, genome.build = NULL)
```

Arguments

`snp.annotation`	Snp annotation dataframe with columns "snpID", "chromosome", and "position". snpID should be a unique integer vector, sorted with respect to chromosome and position.
`ncdf.filename`	The name of the genotype netCDF file to create
`variables`	A character vector containing the names of the variables to create (must be one or more of `c("genotype", "quality", "X", "Y", "rawX", "rawY", "R", "Theta", "BAlleleFreq"`
`n.samples`	The number of samples that will be in the netcdf file.
`precision`	A character value indicating whether floating point numbers should be stored as "double" or "single" precision.
`array.name`	Name of the array, to be stored as an attribute in the netCDF file.
`genome.build`	Genome build used in determining chromosome and position, to be stored as an attribute in the netCDF file.

Details

The function creates a shell netCDF file to which data can subsequently written.

Author(s)

Cathy Laurie

See Also

ncdf, ncdfAddData, ncdfSubset

Examples

```
## Not run:
library(GWASdata)
data(affy_snp_annot)
ncfile <- tempfile()
ncdfCreate(affy_snp_annot, ncfile, variables="genotype", n.samples=5)
file.remove(ncfile)

## End(Not run)
```

NcdfGenotypeReader *Class NcdfGenotypeReader*

Description

The NcdfGenotypeReader class is an extension of the NcdfReader class specific to reading genotype data stored in NetCDF files.

Extends

NcdfReader

Constructor

`NcdfGenotypeReader(filename)`:

`filename` must be the path to a NetCDF file. The NetCDF file must contain the following variables:

- 'snp': a coordinate variable with a unique integer vector of snp ids
- 'chromosome': integer chromosome codes of dimension 'snp'
- 'position': integer position values of dimension 'snp'
- 'sampleID': a unique integer vector of scan ids with dimension 'sample'
- 'genotype': a matrix of bytes with dimensions ('snp','sample'). The byte values must be the number of A alleles : 2=AA, 1=AB, 0=BB.

Default values for chromosome codes are 1-22=autosome, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `autosomeCode`, `XchromCode`, `XYchromCode`, `YchromCode`, and `MchromCode`.

The `NcdfGenotypeReader` constructor creates and returns a NcdfGenotypeReader instance pointing to this file.

Accessors

In the code snippets below, `object` is a NcdfGenotypeReader object. `snp` and `scan` indicate which elements to return along the snp and scan dimensions. They must be integer vectors of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that the entire dimension should be read. If snp and/or is scan omitted, the entire variable is read.

See `NcdfReader` for additional methods.

`nsnp(object)`: The number of SNPs in the NetCDF file.

`nscan(object)`: The number of scans in the NetCDF file.

`getSnpID(object, index)`: A unique integer vector of snp IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getChromosome(object, index, char=FALSE)`: A vector of chromosomes. The optional `index` is a logical or integer vector specifying elements to extract. If `char=FALSE` (default), returns an integer vector. If `char=TRUE`, returns a character vector with elements in (1:22,X,XY,Y,M,U). "U" stands for "Unknown" and is the value given to any chromosome code not falling in the other categories.

`getPosition(object, index)`: An integer vector of base pair positions. The optional `index` is a logical or integer vector specifying elements to extract.

`getScanID(object, index)`: A unique integer vector of scan IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getGenotype(object, snp, scan)`: Extracts genotype values (number of A alleles). The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

`getVariable(object, varname, snp, scan)`: Extracts the contents of the variable `varname`. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`. If the variable is not found in the NetCDF file, returns `NULL`.

`autosomeCode(object)`: Returns the integer codes for the autosomes.

`XchromCode(object)`: Returns the integer code for the X chromosome.

`XYchromCode(object)`: Returns the integer code for the pseudoautosomal region.

`YchromCode(object)`: Returns the integer code for the Y chromosome.

`MchromCode(object)`: Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

NcdfReader, NcdfIntensityReader, GenotypeData, IntensityData

Examples

```
file <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(file)

# dimensions
nsnp(nc)
nscan(nc)

# get snpID and chromosome
snpID <- getSnpID(nc)
chrom <- getChromosome(nc)

# get positions only for chromosome 22
pos22 <- getPosition(nc, index=(chrom == 22))

# get all snps for first scan
geno <- getGenotype(nc, snp=c(1,-1), scan=c(1,1))

# starting at snp 100, get 10 snps for the first 5 scans
geno <- getGenotype(nc, snp=c(100,10), scan=c(1,5))

close(nc)
```

---

ncdfImputedDosage *Create a NetCDF file with imputed dosages - deprecated*

Description

Deprecated - use imputedDosageFile

This function creates a NetCDF file and corresponding annotation for imputed dosages from IMPUTE2, BEAGLE, or MaCH.

Usage

```
ncdfImputedDosage(input.files, ncdf.filename, chromosome,
                  input.type=c("IMPUTE2", "BEAGLE", "MaCH"),
                  input.dosage=FALSE, block.size=5000,
                  snp.annot.filename="dosage.snp.RData",
                  scan.annot.filename="dosage.scan.RData",
                  verbose=TRUE)
```

Arguments

input.files	A character vector of input files. The first file should always be genotypes (either probabilities or dosages). Files for each input type should be as follows: • IMPUTE2: 1) .gens, 2) .samples • BEAGLE: 1) .grobs or .dose, 2) .markers • MaCH: 1) .mlprob or .mldose, 2) .mlinfo, 3) file with columns named "SNP" and "position" giving base pair position of all SNPs
ncdf.filename	Character string with name of output NetCDF file.
chromosome	Chromosome corresponding to the SNPs in the genotype file. Character codes will be mapped to integer values as follows: "X"->23, "XY"->24, "Y"-> 25, "M","MT"->26.
input.type	Format of input files. Accepted file types are "IMPUTE2", "BEAGLE", and "MaCH".
input.dosage	Logical for whether the genotype file (input.files[1]) contains dosages. If FALSE (default), the genotype file is assumed to contain genotype probabilities.
block.size	Number of lines to read at once.
snp.annot.filename	Output .RData file for storing a SnpAnnotationDataFrame.
scan.annot.filename	Output .RData file for storing a ScanAnnotationDataFrame.
verbose	Logical for whether to print progress messages.

Details

Input files can contain either imputed dosages or genotype probabilities, specified by the input.dosage flag. In either case, the NetCDF file will store dosage of the A allele in the "genotype" variable. All SNPs are assumed to be on the same chromosome, which is indicated by the chromosome argument.

SNP and scan annotation are created from the input files and stored in RData format in snp.annot.filename and scan.annot.filename.

Currently supported input file types are IMPUTE2, BEAGLE, and MaCH.

Author(s)

Stephanie Gogarten

References

IMPUTE2: http://mathgen.stats.ox.ac.uk/impute/impute_v2.html

BEAGLE: http://faculty.washington.edu/browning/beagle/beagle.html

MaCH: http://www.sph.umich.edu/csg/abecasis/MACH/tour/imputation.html

See Also

createDataFile, NcdfGenotypeReader, GenotypeData, assocTestRegression

Examples

```
## Not run:
ncfile <- tempfile()
snpfile <- tempfile()
scanfile <- tempfile()

# IMPUTE2
probfile <- system.file("extdata", "imputation", "IMPUTE2", "example.chr22.study.gens",
                        package="GWASdata")
sampfile <- system.file("extdata", "imputation", "IMPUTE2", "example.study.samples",
                        package="GWASdata")
ncdfImputedDosage(input.files=c(probfile, sampfile), ncdf.filename=ncfile, chromosome=22,
                  input.type="IMPUTE2", input.dosage=FALSE,
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)
nc <- NcdfGenotypeReader(ncfile)
scanAnnot <- getobj(scanfile)
snpAnnot <- getobj(snpfile)
genoData <- GenotypeData(nc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
geno <- getGenotype(genoData)
getVariable(genoData, "alleleA")
getVariable(genoData, "alleleB")

# association test with imputed dosages
scanAnnot$status <- sample(0:1, nrow(scanAnnot), replace=TRUE)
genoData <- GenotypeData(nc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
assoc <- assocTestRegression(genoData, outcome="status", model.type="logistic",
                             gene.action.list="additive", dosage=TRUE)
head(assoc)
close(genoData)


# BEAGLE - genotype probabilities
probfile <- system.file("extdata", "imputation", "BEAGLE", "example.hapmap.unphased.bgl.gprobs",
                        package="GWASdata")
markfile <- system.file("extdata", "imputation", "BEAGLE", "hapmap.markers",
                        package="GWASdata")
ncdfImputedDosage(input.files=c(probfile, markfile), ncdf.filename=ncfile, chromosome=22,
                  input.type="BEAGLE", input.dosage=FALSE,
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)

# BEAGLE - dosage
dosefile <- system.file("extdata", "imputation", "BEAGLE", "example.hapmap.unphased.bgl.dose",
                        package="GWASdata")
ncdfImputedDosage(input.files=c(dosefile, markfile), ncdf.filename=ncfile, chromosome=22,
                  input.type="BEAGLE", input.dosage=TRUE,
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)


# MaCH - genotype probabilities
probfile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mlprob",
                        package="GWASdata")
markfile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mlinfo",
```

```
                        package="GWASdata")
posfile <- system.file("extdata", "imputation", "MaCH", "mach1.snp.position",
                        package="GWASdata")
ncdfImputedDosage(input.files=c(probfile, markfile, posfile), ncdf.filename=ncfile, chromosome=22,
                  input.type="MaCH", input.dosage=FALSE,
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)

# MaCH - dosage
dosefile <- system.file("extdata", "imputation", "MaCH", "mach1.out.mldose",
                        package="GWASdata")
ncdfImputedDosage(input.files=c(dosefile, markfile, posfile), ncdf.filename=ncfile, chromosome=22,
                  input.type="MaCH", input.dosage=TRUE,
                  snp.annot.filename=snpfile, scan.annot.filename=scanfile)

unlink(c(ncfile, snpfile, scanfile))

## End(Not run)
```

`NcdfIntensityReader` *Class NcdfIntensityReader*

Description

The NcdfIntensityReader class is an extension of the NcdfReader class specific to reading intensity data stored in NetCDF files.

Extends

NcdfReader

Constructor

`NcdfIntensityReader(filename)`:

`filename` must be the path to a NetCDF file. The NetCDF file must contain the following variables:

- 'snp': a coordinate variable with a unique integer vector of snp ids
- 'chromosome': integer chromosome values of dimension 'snp'
- 'position': integer position values of dimension 'snp'
- 'sampleID': a unique integer vector of scan ids with dimension 'sample'

Default values for chromosome codes are 1-22=autosome, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `autosomeCode`, `XchromCode`, `XYchromCode`, `YchromCode`, and `MchromCode`.

The NetCDF file should also contain at least one of the following variables with dimensions ('snp','sample'):

- 'quality': quality score
- 'X': X intensity

- 'Y': Y intensity
- 'BAlleleFreq': B allele frequency
- 'LogRRatio': Log R Ratio

The `NcdfIntensityReader` constructor creates and returns a NcdfIntensityReader instance pointing to this file.

Accessors

In the code snippets below, `object` is a NcdfIntensityReader object. `snp` and `scan` indicate which elements to return along the snp and scan dimensions. They must be integer vectors of the form (start, count), where start is the index of the first data element to read and count is the number of elements to read. A value of '-1' for count indicates that the entire dimension should be read. If snp and/or is scan omitted, the entire variable is read.

See `NcdfReader` for additional methods.

`nsnp(object)`: The number of SNPs in the NetCDF file.

`nscan(object)`: The number of scans in the NetCDF file.

`getSnpID(object, index)`: A unique integer vector of snp IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getChromosome(object, index, char=FALSE)`: A vector of chromosomes. The optional `index` is a logical or integer vector specifying elements to extract. If `char=FALSE` (default), returns an integer vector. If `char=TRUE`, returns a character vector with elements in (1:22,X,XY,Y,M,U). "U" stands for "Unknown" and is the value given to any chromosome code not falling in the other categories.

`getPosition(object, index)`: An integer vector of base pair positions. The optional `index` is a logical or integer vector specifying elements to extract.

`getScanID(object, index)`: A unique integer vector of scan IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getQuality(object)`: Extracts quality scores. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the NetCDF file contains a variable 'quality'. `hasQuality(object)`:

`getX(object)`: Extracts X intensity. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the NetCDF file contains a variable 'X'. `hasX(object)`:

`getY(object)`: Extracts Y intensity. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the NetCDF file contains a variable 'Y'. `hasY(object)`:

`getBAlleleFreq(object)`: Extracts B allele frequency. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns `TRUE` if the NetCDF file contains a variable 'BAlleleFreq'. `hasBAlleleFreq(object)`:

`getLogRRatio(object)`: Extracts Log R Ratio. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as `NA`.

Returns TRUE if the NetCDF file contains a variable 'LogRRatio'. hasLogRRatio(object):

getVariable(object, varname, snp, scan): Returns the contents of the variable varname. The result is a vector or matrix, depending on the number of dimensions in the returned values. Missing values are represented as NA. If the variable is not found in the NetCDF file, returns NULL.

autosomeCode(object): Returns the integer codes for the autosomes.

XchromCode(object): Returns the integer code for the X chromosome.

XYchromCode(object): Returns the integer code for the pseudoautosomal region.

YchromCode(object): Returns the integer code for the Y chromosome.

McchromCode(object): Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

NcdfReader, NcdfGenotypeReader, GenotypeData, IntensityData

Examples

```
file <- system.file("extdata", "illumina_qxy.nc", package="GWASdata")
nc <- NcdfIntensityReader(file)

# dimensions
nsnp(nc)
nscan(nc)

# get snpID and chromosome
snpID <- getSnpID(nc)
chrom <- getChromosome(nc)

# get positions only for chromosome 22
pos22 <- getPosition(nc, index=(chrom == 22))

# get all snps for first scan
x <- getX(nc, snp=c(1,-1), scan=c(1,1))

# starting at snp 100, get 10 snps for the first 5 scans
x <- getX(nc, snp=c(100,10), scan=c(1,5))

close(nc)
```

NcdfReader *Class NcdfReader*

Description

The NcdfReader class is a wrapper for the **ncdf** library that provides an interface for reading NetCDF files.

Constructor

`NcdfReader(filename)`:

`filename` must be the path to a NetCDF file.

The `NcdfReader` constructor creates and returns a NcdfReader instance pointing to this file.

Accessors

In the code snippets below, `object` is a NcdfReader object.

`getVariable(object, varname, start, count)`: Returns the contents of the variable `varname`.

- `start` is a vector of integers indicating where to start reading values. The length of this vector must equal the number of dimensions the variable has. If not specified, reading starts at the beginning of the file (1,1,...).
- `count` is a vector of integers indicating the count of values to read along each dimension. The length of this vector must equal the number of dimensions the variable has. If not specified and the variable does NOT have an unlimited dimension, the entire variable is read. As a special case, the value "-1" indicates that all entries along that dimension should be read.

The result is a vector, matrix, or array, depending on the number of dimensions in the returned values. Missing values (specified by a "missing_value" attribute, see `set.missval.ncdf`) are represented as `NA`. If the variable is not found in the NetCDF file, returns `NULL`.

`getVariableNames(object)`: Returns names of variables in the NetCDF file.

`getDimension(object, varname)`: Returns dimension for NetCDF variable `varname`.

`getDimensionNames(object, varname)`: Returns names of dimensions in the NetCDF file. If `varname` is provided, returns dimension names for NetCDF variable `varname`.

`getAttribute(object, attname, varname)`: Returns the attribute `attname` associated with the variable `varname`. If `varname` is not specified, `attname` is assumed to be a global attribute.

`hasCoordVariable(object, varname)`: Returns `TRUE` if `varname` is a coordinate variable (a variable with the same name as a dimension).

`hasVariable(object, varname)`: Returns `TRUE` if `varname` is a variable in the NetCDF file (including coordinate variables).

Standard Generic Methods

In the code snippets below, `object` is a NcdfReader object.

`open(object)`: Opens a connection to a NetCDF file.

`close(object)`: Closes the connection to a NetCDF file.

`show(object)`: Summarizes the contents of a NetCDF file.

Author(s)

Stephanie Gogarten

See Also

ncdf, `NcdfGenotypeReader`, `NcdfIntensityReader`

Examples

```
file <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc <- NcdfReader(file)

getDimensionNames(nc)
getVariableNames(nc)

hasVariable(nc, "genotype")
geno <- getVariable(nc, "genotype", start=c(1,1), count=c(10,10))

close(nc)
```

---

ncdfSetMissingGenotypes

Write a new netCDF or GDS file, setting certain SNPs to missing - deprecated

---

Description

Deprecated - use `setMissingGenotypes`

`ncdfSetMissingGenotypes` copies an existing netCDF genotype file to a new one, setting SNPs in specified regions to missing. `gdsSetMissingGenotypes` copies an existing GDS genotype file to a new one, setting SNPs in specified regions to missing.

Usage

```
ncdfSetMissingGenotypes(parent.file, new.file, regions,
                        sample.include=NULL, verbose=TRUE)
gdsSetMissingGenotypes(parent.file, new.file, regions,
                       sample.include=NULL,
                       zipflag="ZIP.max", verbose=TRUE)
```

Arguments

parent.file	Name of the parent file
new.file	Name of the new file
regions	Data.frame of chromosome regions with columns "scanID", "chromosome", "left.base", "right.ba
sample.include	Vector of sampleIDs to include in new.file
zipflag	the compression format for the GDS file, one of "", "ZIP", "ZIP.fast", "ZIP.default", or "ZIP.max"
verbose	Logical value specifying whether to show progress information.

Details

ncdfSetMissingGenotypes and gdsSetMissingGenotypes remove chromosome regions by setting SNPs that fall within the anomaly regions to NA (i.e., the missing value in the netCDF/GDS file). Optionally, entire samples may be excluded from the netCDF/GDS file as well: if the sample.include argument is given, only the scanIDs in this vector will be written to the new file, so the sample dimension will be length(sample.include).

For regions with whole.chrom=TRUE, the entire chromosome will be set to NA for that sample. For other regions, only the region between left.base and right.base will be set to NA.

Author(s)

Stephanie Gogarten

See Also

ncdfSubset, anomSegStats for chromosome anomaly regions

Examples

```
## Not run:
ncfile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(ncfile)
sample.sel <- getScanID(nc, index=1:10)
close(nc)

regions <- data.frame("scanID"=sample.sel[1:3], "chromosome"=c(21,22,23),
  "left.base"=c(14000000, 30000000, NA), "right.base"=c(28000000, 450000000, NA),
  whole.chrom=c(FALSE, FALSE, TRUE))

newnc <- tempfile()
ncdfSetMissingGenotypes(ncfile, newnc, regions, sample.include=sample.sel)
file.remove(newnc)

## End(Not run)
```

ncdfSubset *Write a subset of data in a netCDF file to a new netCDF file*

Description

ncdfSubset takes a subset of data (snps and samples) from a netCDF file and write it to a new netCDF file. ncdfSubsetCheck checks that a netCDF file is the desired subset of another netCDF file.

Usage

```
ncdfSubset(parent.ncdf, sub.ncdf,
           sample.include=NULL, snp.include=NULL,
           verbose=TRUE)

ncdfSubsetCheck(parent.ncdf, sub.ncdf,
                sample.include=NULL, snp.include=NULL,
                verbose=TRUE)
```

Arguments

parent.ncdf	Name of the parent netCDF file
sub.ncdf	Name of the subset netCDF file
sample.include	Vector of sampleIDs to include in sub.ncdf
snp.include	Vector of snpIDs to include in sub.ncdf
verbose	Logical value specifying whether to show progress information.

Details

ncdfSubset can select a subset of snps for all samples by setting snp.include, a subset of samples for all snps by setting sample.include, or a subset of snps and samples with both arguments.

Author(s)

Cathy Laurie, Stephanie Gogarten

See Also

ncdf, createDataFile

Examples

```
ncfile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(ncfile)
sample.sel <- getScanID(nc, index=1:10)
snp.sel <- getSnpID(nc, index=1:100)
close(nc)
```

```
subnc <- tempfile()
ncdfSubset(ncfile, subnc, sample.include=sample.sel, snp.include=snp.sel)
ncdfSubsetCheck(ncfile, subnc, sample.include=sample.sel, snp.include=snp.sel)
file.remove(subnc)
```

`pasteSorted` *Paste two vectors sorted pairwise*

Description

Read a configuration file

Usage

```
pasteSorted(a, b, sep="/")
```

Arguments

a	vector 1
b	vector 2
sep	a character string to separate the terms.

Value

A character vector of the concatenated values, sorted pairwise.

Author(s)

Stephanie Gogarten

See Also

`paste`

Examples

```
a <- c("A","C","G","T")
b <- c("C","A","T","G")
pasteSorted(a,b)
```

pcaSnpFilters *Regions of SNP-PC correlation to filter for Principal Component Analysis*

Description

Base positions for the LCT (2q21), HLA (including MHC), and inversion (8p23, 17q21.31) regions from the GRCh36/hg18 and GRCh37/hg19 genome builds.

Usage

```
pcaSnpFilters.hg18
pcaSnpFilters.hg19
```

Format

A data.frame with the following columns.

`chrom` chromsome

`start.base` starting base position of region

`end.base` ending base position of region

`comment` description of the region

Details

These regions result in high SNP-PC correlation if they are included in Principal Component Analysis (PCA). The `pcaSnpFilters` datasets can be used to filter SNPs prior to running PCA to avoid correlations.

Source

UCSC genome browser (http://genome.ucsc.edu).

References

Novembre, John et al. (2008), Genes mirror geography within Europe. Nature, 456: 98-101. doi:10.1038/nature07331

See Also

`snpCorrelationPlot`, `SNPRelate`

Examples

```
data(pcaSnpFilters.hg18)
data(pcaSnpFilters.hg19)
```

pedigreeCheck *Testing for internal consistency of pedigrees*

Description

Find inconsistencies within pedigrees.

Usage

```
pedigreeCheck(pedigree)
```

Arguments

pedigree	A dataframe containing the pedigree information for the samples to be examined with columns labeled "family", "individ", "mother", "father" and "sex" containing the identifiers of the family, individual, individual's mother, individual's father and individual's sex (coded as "M" or "F") . Identifiers can be integer, numeric or character but identifiers for mother and father for founders are assumed to be 0.

Details

The function pedigreeCheck finds any of a number of possible errors and inconsistencies within pedigree data. If no problems are encountered, the output is NULL. If problems are encountered, output contains information for the errors encountered (a sub-list of the output values described below) and the following message is printed: "All row numbers refer to rows in the full pedigree (not just within a family). Correct current problems and rerun pedigreeCheck. There may be additional problems not investigated because of the current problems."

Value

The output for pedigreeCheck is NULL or a sub-list of the following:

family.missing.rows	A vector of integers containing the row positions of entries in the full pedigree where family id's are missing (NA) or blank
individ.missing_or_0.rows	A vector of integers containing the row positions of entries in the full pedigree where individual id's are missing (NA), blank, or 0
father.missing.rows	A vector of integers containing the row positions of entries in the full pedigree where father id's are missing (NA) or blank
mother.missing.rows	A vector of integers containing the row positions of entries in the full pedigree where mother id's are missing (NA) or blank
sexcode.error.rows	A vector of integers containing the row positions of entries in the full pedigree where the 'sex' variable is mis-coded

`both.mother.father`
: A data.frame with the variables 'family','parentID','mother.row',and 'father.row' where 'family' = family identifier, 'parentID' = identifier of parent that appears as both mother and father, 'father.row' = row positions(s) in full pedigree in which parent appears as father, and 'mother.row' = row position(s) in full pedigree in which parent appears as mother (if mutliple rows, row numbers are concatenated with separator = ';')

`parent.no.individ.entry`
: A data.frame with the variables 'row.num', 'family', 'no_individ_entry', and 'parentID', where 'row.num' = row position of entry in the full pedigree where mother and/or father IDs are not included in the pedigree, 'family' = family identifier, 'no_individ_entry' has values 'father', 'mother' or 'both' indicating which parent is not in the pedigree, and 'parentID' = the identifier(s) for individuals not in the pedigree (if more than one, identifiers are concatenated with separator =';')

`unknown.parent.rows`
: A data.frame with variables 'row.num' = row position in full pedigree where one parent is known and one parent is unknown and 'family' = family identifier.

`duplicates`
: A data.frame with variables 'family' = family identifier, 'individ' = individual identifier, 'copies' = number of copies of individual and 'match'= T/F depending upon whether all copies have identical pedigree information

`one.person.fams`
: A data.frame identifying singeltons (one person families) with variables 'family' = family identifier and 'founder' = T/F depending up whether the singleton is a founder or not

`mismatch.sex`
: A data.frame with variables 'family' = family identifier and 'individ' = individual identifier for individuals that occur as mothers but sex is "M" or occur as fathers but sex is "F"

`impossible.related.rows`
: A list where each entry in the list contains a set of row positions in the full pedigree which together indicate impossible relationships: where either a child is mother of self or an individual is both child and mother of the same person. Names of list entries are associated family identifiers.

`subfamilies.ident`
: A data.frame with variables 'family' = family identifier, "subfamily" = subfamily identifier within family, and 'individ' = individual identifier of members of identified sub-family.

If no inconsistencies are found, the output is `NULL`.

Note

All row numbers in output refer to row positions in the full pedigree (not just within family). User should correct current problems and rerun pedigreeCheck. There may be additional problems not investigated because of the current problems.

Author(s)

Cecelia Laurie

See Also

pedigreeDeleteDuplicates, pedigreePairwiseRelatedness

Examples

```
#basic errors
family <- c("a","a","a","b","b","c","")
individ <- c("A","B","C","A","B",0,"")
mother <- c("B","C",0,0,0,NA,0)
father <- c("C","D",0,0,"",0,"D")
sex <- c("F","2","M","F","F","M","F")
samp <- data.frame(family, individ, mother,father,sex,stringsAsFactors=FALSE)
pedigreeCheck(samp)
# there are other problems not investigated since
#    the above are basic problems to be cleared up first

## duplicates, both.mother.father, parent.no.individ.entry
family <- c("b","b","b","b","c","c",rep("d",5))
individ <- c("A","B","C","A","B","B",1:5)
mother <- c("B",0,0,"D",0,0,0,0,1,2,1)
father <- c("C",0,0,"C",0,0,0,0,2,1,2)
sex <- c("F","F","M","M","F","F","F","M","F","F","M")
samp <- data.frame(family, individ, mother,father,sex,stringsAsFactors=FALSE)
pedigreeCheck(samp)
# there are other problems (such as mismatch.sex) but not investigated
#     directly because already had both.mother.father inconsistency

# parent.no.individ.entry, one.person.fams, unknown.parent.rows,
#    mismatch.sex,impossible.related.rows
family <- c(1,1,1,2,2,2,3,4,4,4,5,5,5,5,6,6,6)
individ <- c(1,2,3,1,2,3,1,1,3,2,1,2,3,4,1,2,3)
mother <- c(2,0,1,2,1,0,1,2,0,2,2,4,0,0,2,1,0)
father <- c(3,0,3,0,3,0,2,3,1,0,3,1,0,0,3,3,0)
sex <- c("F","F","M","F","F","M","F","F","F","F","M","F","M","F","F","M","F")
samp <- data.frame(family, individ,mother,father,sex,stringsAsFactors=FALSE)
pedigreeCheck(samp)
# mismatch.sex and impossible.related.rows are only investigated
#      for families where there are no other inconsistencies

## subfamilies.ident
family <- rep(1,12)
individ <- 1:12
mother <- c(0,0,2,2,0,0,5,0,7,0,0,10)
father <- c(0,0,1,1,0,0,6,0,8,0,0,11)
sex <- c("M",rep("F",4),"M","F","M","M","F","M","M")
samp <- data.frame(family,individ,mother,father,sex,stringsAsFactors=FALSE)
pedigreeCheck(samp)
# subfamilies.ident is only investigated for families
#     where there are no other inconsistencies
```

pedigreeDeleteDuplicates *Remove duplicates from a pedigree*

Description

pedigreeDeleteDuplicates removes duplicates from a pedigree.

Usage

```
pedigreeDeleteDuplicates(pedigree, duplicates)
```

Arguments

pedigree	A dataframe containing the pedigree information for the samples to be examined with columns labeled "family", "individ", "mother", "father" and "sex" containing the identifiers of the family, individual, individual's mother, individual's father and individual's sex (coded as "M" or "F") .
duplicates	dataframe with columns "family" (family id) and "individ" (individual id).

Details

The output of pedigreeCheck can be provided to pedigreeDeleteDuplicates in order to generate a new pedigree with duplicates removed.

Value

The output of pedigreeDeleteDuplicates is a pedigree identical to pedigree, but with duplicates removed.

Author(s)

Cecelia Laurie

See Also

pedigreeCheck, pedigreePairwiseRelatedness

Examples

```
family <- c(1,1,1,1,2,2,2,2)
individ <- c(1,2,3,3,4,5,6,6)
mother <- c(0,0,1,1,0,0,4,4)
father <- c(0,0,2,2,0,0,5,5)
sex <- c("F","M","F","F","F","F","M","M")
pedigree <- data.frame(family, individ, mother, father, sex, stringsAsFactors=FALSE)
duplicates <- pedigreeCheck(pedigree)$duplicates
pedigree.no.dups <- pedigreeDeleteDuplicates(pedigree, duplicates)
```

pedigreeMaxUnrelated *Find a maximal set of unrelated individuals in a subset of a pedigree.*

Description

Given a full pedigree (with no duplicates and no one-person families), this function finds a maximal set of unrelated individuals in a specified subset of the pedigree. This is done family by family. The full pedigree is checked for inconsistencies and an error message is given if inconsistencies are found (see pedigreeCheck). Maximal sets are not unique; there is an option for the user to identify preference(s) in the choice of individuals.

Usage

```
pedigreeMaxUnrelated(pedigree, pref = NULL)
```

Arguments

pedigree	A dataframe containing the full pedigree with columns 'family', 'individ', 'mother', 'father', 'sex', and 'selset'. The variables 'family', 'individ', 'mother', 'father' contain the identifiers for family, individual, individual's mother and individual's father. Identifiers can be integer, numeric or character but identifiers for mother and father for founders are assumed to be 0. The variable 'sex' contains the individual's sex (coded as "M" or "F"). The varible 'selset' is coded as 1 = if individual is in the subset of interest and 0 otherwise. The dataframe can contain an optional variable indicating preferences for choosing individuals. See the item pref below.
pref	pref = the name of the (optional) preference column in samp. Preferences can be layered. This variable must have integer or numeric values greater than or equal to 1 where a lower value indicates higher preference. If pref is missing, the default is to prefer choosing founders.

Details

Commonly used for selecting a maximal unrelated set of genotyped individuals from a pedigree ('selset' = 1 if individual is genotyped and 0 otherwise).

An example of the use of a layered preference variable: if one wanted to prefer cases over controls and then prefer founders, the preference variable would = 1 for cases, 2 = founder, 3 = otherwise.

Value

A dataframe with variables 'family' = family identifier and 'Individ' = individual identifier of individuals in the maximal unrelated set.

Note

Since pedigreeMaxUnrelated does not accept one-person families included in the input pedigree, to get a complete maximal set of unrelated individuals from a specified subset of the pedigree, the user will need to append to the output from the function the one-person family (singleton) individuals from the specified subset.

Author(s)

Cecelia Laurie

See Also

pedigreeCheck, pedigreePairwiseRelatedness

Examples

```
## Example set 1
family <- rep("A",8)
individ <- c("a","b","c","d","e","f","g","h")
mother <- c(0,"a","b",0,"f",0,0,"f")
father <- c(0,"d","e",0,"g",0,0,"g")
sex <- c(rep("F",3),"M","M","F","M","F")
pedigree <- data.frame(family, individ, mother, father, sex, stringsAsFactors=FALSE)

## preference default (i.e. choose founders if possible)
pedigree$selset <- 1  # all selected
pedigreeMaxUnrelated(pedigree)   # chose the founders
#  family Individ
#1      A       a
#2      A       d
#3      A       f
#4      A       g

sel <- is.element(pedigree$individ,c("a","f","g"))
pedigree$selset[sel] <- 0  #only one founder d in desired subset

#  default preference of founders
pedigreeMaxUnrelated(pedigree)
#  family Individ
#1      A       d    #founder
#2      A       e

## preference choice
pedigree$pref <- 2
sel2 <- is.element(pedigree$individ, c("c","h")) # preferred choices
pedigree$pref[sel2] <- 1
pedigreeMaxUnrelated(pedigree,pref="pref")
#  family Individ
#1      A       h
#2      A       b

## add preference layer of secondary choice of founders
```

```
pedigree$pref <- 3
sel2 <- pedigree$mother==0 & pedigree$father==0
sel1 <- is.element(pedigree$individ, c("c","h"))
pedigree$pref[sel2] <- 2
pedigree$pref[sel1] <- 1
pedigreeMaxUnrelated(pedigree,pref="pref")
#  family Individ
#1      A       h   #top pref
#2      A       d   #founder
#Note that the other top preference c is related to everyone so not chosen

## Example Set 2
family <- c(1,1,1,1,2,2,2,2,2)
individ <- c(2,1,3,4,"A5","A6","A7","A8","A9")
mother <- c(3,3,0,0,0,0,"A5","A5",0)
father <- c(4,4,0,0,0,0,"A6","A9",0)
sex <- c("F","M","F","M","F","M","M","M","M")
pedigree <- data.frame(family, individ, mother, father, sex, stringsAsFactors=FALSE)
pedigree$selset <- 1
pedigree$selset[is.element(pedigree$individ, c("A5",4))] <- 0
pedigree$pref <- 2
pedigree$pref[is.element(pedigree$individ,c("A8","A7"))] <- 1
pedigreeMaxUnrelated(pedigree,pref="pref")
#  family Individ
#1      1       2
#2      2      A6
#3      2      A8
# NOTE: in using the pref option there is NO preference for family 1
# so will select one unrelated from family 1:
# individual 2 is selected since it is first in selset to be listed in pedigree

pedigree$pref <- 2
pedigree$pref[is.element(pedigree$individ,c("A8","A7"))] <- 1
sel <- pedigree$family==1 & pedigree$mother==0 & pedigree$father==0 #founders
pedigree$pref[sel] <- 1
pedigreeMaxUnrelated(pedigree,pref="pref")
#  family Individ
#1      1       3
#2      2      A6
#3      2      A8
```

---

pedigreePairwiseRelatedness

Assign relatedness from pedigree data

Description

This function assigns relationships from pedigree data. Output includes the theoretical pairwise kinship coefficients.

Usage

```
pedigreePairwiseRelatedness(pedigree)
```

Arguments

pedigree	A dataframe containing the pedigree information for the samples to be examined with columns labeled "family", "individ", "mother", "father" and "sex" containing the identifiers for family, individual, individual's mother, individual's father and individual's sex (coded as "M" or "F") . Identifiers can be integer, numeric or character but identifiers for mother and father for founders are assumed to be 0. Error messages are returned for pedigree inconsistencies. See pedigreeCheck

Details

Assigns relationships between individuals in a pedigree, including "U" = unrelated, "PO" = parent/offspring, "FS" = full siblings, "HS" = half siblings, "Av" = avuncular, "GpGc" = grandparent-grandchild, and "FC" = first cousins, among others).

Relatedness is not calculated for inbred families but kinship coefficients are.

Value

A list with the following components:

inbred.fam	A vector of id's of families with inbreeding (relationships are not assigned).
inbred.KC	A dataframe for inbred families with columns "Individ1","Individ2", "kinship" and "family" containing the id's of the pair of individuals, kinship coefficient and family id.
relativeprs	A dataframe with columns "Individ1", "Individ2", "relation", "kinship" and "family" containing the id's of the pair of individuals, the relationship between the individuals if closely related (possible values are "U" = unrelated, "PO" = parent/offspring, "FS" = full siblings, "HS" = half siblings, "Av" = avuncular, "GpGc" = grandparent-grandchild, and "FC" = first cousins, among others), kinship coefficient and family id.

Author(s)

Cecelia Laurie

See Also

pedigreeCheck, pedigreeMaxUnrelated

Examples

```
family <- c(1,1,1,1,2,2,2,2,2,2,2)
individ <- c(1,2,3,4,5,6,7,8,9,10,11)
mother <- c(0,0,1,1,0,0,5,5,0,0,10)
father <- c(0,0,2,2,0,0,6,9,0,0,7)
sex <- c("F","M","F","F","F","M","M","M","M","F","F")
```

```
pedigree <- data.frame(family, individ, mother, father, sex, stringsAsFactors=FALSE)
pedigreePairwiseRelatedness(pedigree)

# inbred family
family <- rep(2,7)
individ <- paste("I",c(1,2,3,4,5,6,7),sep="")
mother <- c(0,0,0,"I1","I1","I3","I5")
father <- c(0,0,0,"I2","I2","I4","I4")
sex <- c("F","M","F","M","F","F","F")
samp2 <- data.frame(family, individ, mother, father, sex, stringsAsFactors=FALSE)
pedigreePairwiseRelatedness(samp2)
```

`plinkToNcdf` *Create a netCDF file and annotation suitable for use in GWASTools from PLINK files*

Description

`plinkToNcdf` creates a netCDF file and scan and SNP annotation objects from a set of ped and map files.

Usage

```
plinkToNcdf(pedFile, mapFile, nSamples,
  ncdfFile, snpAnnotFile, scanAnnotFile,
  ncdfXchromCode=23, ncdfXYchromCode=24, ncdfYchromCode=25,
  ncdfMchromCode=26, ncdfUchromCode=27,
  pedMissingCode=0, verbose=TRUE)
```

Arguments

`pedFile`	PLINK ped file.
`mapFile`	PLINK map file. Columns should be chromosome, rsID, map distance (not used, but included in output annotation), and base-pair position. If this is an extended map file (.bim), columns 5 and 6 will be used to encode allele A and allele B.
`nSamples`	Number of samples in the ped file.
`ncdfFile`	Output netCDF file.
`snpAnnotFile`	Output .RData file for storing a `SnpAnnotationDataFrame`.
`scanAnnotFile`	Output .RData file for storing a `ScanAnnotationDataFrame`.
`ncdfXchromCode`	Integer value used to represent the X chromosome in the netCDF file. Values of "X" or "23" in the map file are converted to this code.
`ncdfXYchromCode`	Integer value used to represent the pseudoautosomal region of the X and Y chromsomes in the netCDF file. Values of "XY" or "25" in the map file are converted to this code.

ncdfYchromCode	Integer value used to represent the Y chromosome in the netCDF file. Values of "Y" or "24" in the map file are converted to this code.
ncdfMchromCode	Integer value used to represent mitochondrial SNPs in the netCDF file. Values of "MT" or "26" in the map file are converted to this code.
ncdfUchromCode	Integer value used to represent unknown chromosome in the netCDF file. Any values in the map file not in (1:26, "X", "Y", "XY", "MT") are converted to this code.
pedMissingCode	Missing genotype code in the ped file.
verbose	logical for whether to show progress information.

Details

The netCDF file stores genotype data in byte format, so the PLINK genotype is converted to number of A alleles (0, 1, 2, or missing). The definitions of A and B alleles may be provided in the map file (column 5=A, column 6=B). Otherwise, A and B definitions will be based on the order alleles are encountered in the ped file. (Note that converting between ped/map format and bed/bim/fam format in PLINK will not always preserve the order of chromosomes, so use caution when matching a bim file to a ped file!)

The first six columns of the ped file will be converted to a ScanAnnotationDataFrame. If the Individual ID (second column of the ped file) contains unique integers, then this column will be used for scanID. Otherwise, an integer vector of scanID will be generated as `1:nSamples`. This ID is used to index scans in the netCDF file.

The map file will be converted to a SnpAnnotationDataFrame. This SNP annotation will include the definitions of A and B alleles in the netCDF file (either as provided or determined from the data as described above). A unique integer snpID will be generated for each SNP, which is used to index SNPs in the netCDF file.

Note that the default values of `ncdfXYchromCode=24`, `ncdfYchromCode=25`, and `ncdfUchromCode=27` correspond to the default chromosome codes for NcdfGenotypeReader and SnpAnnotationDataFrame, and are different from the values used by PLINK (Y=24, XY=25, U=0). If the netCDF file is created with different chromosome codes by specifying these arguments, one must also specify the chromosome codes when opening the file, e.g. `NcdfGenotypeReader(ncdfFile, XYchromCode=25, YchromCode=24)`.

`nSamples` is used to allocate space in the netCDF file. A warning will be issued if the number of lines read in the ped file is different from this number.

Author(s)

Stephanie Gogarten

References

Please see http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml#ped for more information on PLINK files.

See Also

plinkWrite, plinkCheck

Examples

```
library(GWASdata)
pedfile <- system.file("extdata", "illumina_subj.ped", package="GWASdata")
mapfile <- system.file("extdata", "illumina_subj.map", package="GWASdata")
ncfile <- tempfile()
scanfile <- tempfile()
snpfile <- tempfile()
plinkToNcdf(pedfile, mapfile, nSamples=43, ncdfFile=ncfile,
   snpAnnotFile=snpfile, scanAnnotFile=scanfile)

nc <- NcdfGenotypeReader(ncfile)
scanAnnot <- getobj(scanfile)
snpAnnot <- getobj(snpfile)
genoData <- GenotypeData(nc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
prefix <- sub(".ped", "", pedfile, fixed=TRUE)
log <- tempfile()
stopifnot(plinkCheck(genoData, prefix, log))
close(genoData)

# provide allele coding with extended map file
# .bim might have SNPs in different order than .map
bimfile <- system.file("extdata", "illumina_subj.bim", package="GWASdata")
bim <- read.table(bimfile, as.is=TRUE, header=FALSE)
map <- read.table(mapfile, as.is=TRUE, header=FALSE)
snp.match <- match(map[,2], bim[,2])
map <- cbind(map, bim[snp.match, 5:6])
mapfile.ext <- tempfile()
write.table(map, file=mapfile.ext, quote=FALSE, row.names=FALSE, col.names=FALSE)
# use chromosome codes that match PLINK
plinkToNcdf(pedfile, mapfile, nSamples=43, ncdfFile=ncfile,
   snpAnnotFile=snpfile, scanAnnotFile=scanfile,
   ncdfYchromCode=24, ncdfXYchromCode=25)

# must specify different chromosome codes in NcdfGenotypeReader
# appending "L" ensures the codes are integers, as required
nc <- NcdfGenotypeReader(ncfile, YchromCode=24L, XYchromCode=25L)
scanAnnot <- getobj(scanfile)
snpAnnot <- getobj(snpfile)
genoData <- GenotypeData(nc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
stopifnot(plinkCheck(genoData, prefix, log))
close(genoData)

file.remove(ncfile, scanfile, snpfile, log, mapfile.ext)
```

---

plinkUtils *Utilities to create and check PLINK files*

Description

`plinkWrite` creates ped and map format files (used by PLINK) from a `GenotypeData` object. `plinkCheck` checks whether a set of ped and map files has identical data to a `GenotypeData` object.

Usage

```
plinkWrite(genoData, pedFile="testPlink", family.col="family",
  individual.col="scanID", father.col="father", mother.col="mother",
  phenotype.col=NULL,
  rs.col="rsID", mapdist.col=NULL, scan.exclude=NULL,
  scan.chromosome.filter=NULL, blockSize=100, verbose=TRUE)

plinkCheck(genoData, pedFile, logFile="plinkCheck.txt", family.col="family",
  individual.col="scanID", father.col="father", mother.col="mother",
  phenotype.col=NULL,
  rs.col="rsID", map.alt=NULL, check.parents=TRUE, check.sex=TRUE,
  scan.exclude=NULL, scan.chromosome.filter=NULL, verbose=TRUE)
```

Arguments

genoData	A GenotypeData object with scan and SNP annotation.
pedFile	prefix for PLINK files (pedFile.ped, pedFile.map)
logFile	Name of the output file to log the results of plinkCheck
family.col	name of the column in the scan annotation that contains family ID of the sample
individual.col	name of the column in the scan annotation that contains individual ID of the sample
father.col	name of the column in the scan annotation that contains father ID of the sample
mother.col	name of the column in the scan annotation that contains mother ID of the sample
phenotype.col	name of the column in the scan annotation that contains phenotype variable (e.g. case control statue) of the sample
rs.col	name of the column in the SNP annotation that contains rs ID (or some other ID) for the SNP
mapdist.col	name of the column in the SNP annotation that contains genetic distance in Morgans for the SNP
map.alt	data frame with alternate SNP mapping for genoData to PLINK. If not NULL, this annotation will be used to compare SNP information to the PLINK file, rather than the default conversion from the SNP annotation embedded in genoData. Columns should include "snpID", "rsID", "chromosome", "position".
check.parents	logical for whether to check the father and mother columns
check.sex	logical for whether to check the sex column
scan.exclude	vector of scanIDs to exclude from PLINK file
scan.chromosome.filter	a logical matrix that can be used to zero out (set to missing) some chromosomes, some scans, or some specific scan-chromosome pairs. Entries should be TRUE if that scan-chromosome pair should have data in the PLINK file, FALSE if not. The number of rows must be equal to the number of scans in genoData. The column labels must be in the set ("1":"22", "X", "XY", "Y", "M", "U").
blockSize	Number of samples to read from genoData at a time
verbose	logical for whether to show progress information.

Details

If "alleleA" and "alleleB" columns are not found in the SNP annotation of genoData, genotypes are written as "A A", "A B", "B B" (or "0 0" for missing data).

If phenotype.col=NULL, plinkWrite will use "-9" for writing phenotype data and plinkCheck will omit checking this column.

If mapdist.col=NULL, plinkWrite will use "0" for writing this column in the map file and plinkCheck will omit checking this column.

plinkCheck first reads the map file and checks for SNP mismatches (chromosome, rsID, and/or position). Any mismatches are written to logFile. plinkCheck then reads the ped file line by line, recording all mismatches in logFile. SNPs and sample order is not required to be the same as in genoData. In the case of genotype mismatches, for each sample the log file output gives the position of the first mismatched SNP in the PLINK file, as well as the genotypes of the first six mismatched SNPs (which may not be consecutive).

These utilities convert between chromosome coding in GenotypeData, which by default is 24=XY, 25=Y, and PLINK chromosome coding, which is 24=Y, 25=X.

Larger blockSize will improve speed but will require more RAM.

Value

plinkCheck returns TRUE if the PLINK files contain identical data to genoData, and FALSE if a mismatch is encountered.

Author(s)

Stephanie Gogarten, Tushar Bhangale

References

Please see http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml#ped for more information on the ped and map files.

See Also

plinkToNcdf

Examples

```
library(GWASdata)
ncfile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
data(illuminaSnpADF, illuminaScanADF)
genoData <- GenotypeData(NcdfGenotypeReader(ncfile),
  scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)

pedfile <- tempfile()
plinkWrite(genoData, pedfile)
logfile <- tempfile()
plinkCheck(genoData, pedfile, logfile)
```

```
# exclude samples
plinkWrite(genoData, pedfile, scan.exclude=c(281, 283),
  blockSize=10)
plinkCheck(genoData, pedfile, logfile)
readLines(logfile)
#samples not found in Ped:
#281
#283

close(genoData)
unlink(c(logfile, paste(pedfile, "*", sep=".")))
```

`pseudoautoIntensityPlot` *Plot B Allele Frequency and Log R Ratio for the X and Y chromosomes, overlaying XY SNPs*

Description

This function plots X, Y and pseudoautosomal SNPs on BAF/LRR plots.

Usage

```
pseudoautoIntensityPlot(intenData, scan.ids, main=NULL,
  plotY=FALSE, hg.build=c("hg18", "hg19"),
  snp.exclude = NULL, cex=0.5, ...)
```

Arguments

`scan.ids`	A vector containing the sample indices of the plots.
`intenData`	`IntensityData` object, must contain 'BAlleleFreq' and 'LogRRatio'
`main`	A character vector containing the titles to be used for each plot. If `NULL` then the title will be the sample number and the chromosome.
`plotY`	If `plotY` is `TRUE`, the Y chromosome will be plotted in addition to X.
`hg.build`	Human genome bulid number
`snp.exclude`	An integer vector giving the IDs of SNPs to exclude from the plot.
`cex`	cex value for points on the plots
`...`	Other parameters to be passed directly to `plot`.

Details

The pseudoautosomal regions are highlighted on the plots (PAR1 and PAR2 in gray, XTR in yellow), and the X, Y, and XY SNPs are plotted in different colors. The base positions for these regions depend on genome build (`hg.build`). Currently hg18 and hg19 are supported.

By default the output is a 2-panel plot with LRR and BAF for the X chromosome. if `plotY` is `TRUE`, the output is a 4-panel plot with the Y chromosome plotted as well.

Author(s)

Caitlin McHugh

References

Ross, Mark. T. et al. (2005), The DNA sequence of the human X chromosome. Nature, 434: 325-337. doi:10.1038/nature03440

Mumm, S., Molini, B., Terrell, J., Srivastava, A., and Schlessinger, D. (1997), Evolutionary features of the 4-Mb Xq21.3 XY homology region revealed by a map at 60-kb resolution. Genome Res. 7: 307-314.

See Also

pseudoautosomal, IntensityData, GenotypeData, BAFfromGenotypes

Examples

```
library(GWASdata)
data(illuminaScanADF)
blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
blgds <- GdsIntensityReader(blfile)
intenData <-  IntensityData(blgds, scanAnnot=illuminaScanADF)

scanID <- getScanID(illuminaScanADF, index=1)
pseudoautoIntensityPlot(intenData=intenData, scan.ids=scanID)
close(intenData)
```

---

pseudoautosomal *Pseudoautosomal region base positions*

Description

Pseudoautosomal region (XTR, PAR1, PAR2) base positions for the X and Y chromsosomes from the GRCh36/hg18 and GRCh37/hg19 genome builds.

Usage

```
pseudoautosomal.hg18
pseudoautosomal.hg19
```

Format

A data.frame with the following columns.

`chrom` chromosome (X or Y)2

`region` region (XTR, PAR1, or PAR2)

`start.base` starting base position of region

`end.base` ending base position of region

Details

The XTR region on X is defined as DXS1217 to DXS3. The XTR region on Y is defined as SY20 to DXYS1.

Source

UCSC genome browser (http://genome.ucsc.edu).

References

Ross, Mark. T. et al. (2005), The DNA sequence of the human X chromosome. Nature, 434: 325-337. doi:10.1038/nature03440

Mumm, S., Molini, B., Terrell, J., Srivastava, A., and Schlessinger, D. (1997), Evolutionary features of the 4-Mb Xq21.3 XY homology region revealed by a map at 60-kb resolution. Genome Res. 7: 307-314.

Examples

```
data(pseudoautosomal.hg18)
data(pseudoautosomal.hg19)
```

---

qqPlot *QQ plot for genome wide assocation studies*

Description

Generates a Quantile-Quantile plot for -log10 p-values from genome wide association tests.

Usage

```
qqPlot(pval, truncate = FALSE, ylim = NULL, thinThreshold = NULL, ...)
```

Arguments

pval	Vector of p-values
truncate	Either a logical value indicating whether the y-axis should be truncted to the same range as the x-axis, or a numeric value indicating where to truncate the y-axis. See details.
ylim	Limits for the y axis. Ignored if truncate=TRUE or truncate is numeric.
thinThreshold	if not NULL, -log10(pval) threshold for thinning points.
...	Other parameters to be passed directly to plot.

Details

The function generates a Quantile-Quantile plot of p-values on a -log10 scale, with the option of truncating the y-axis to the range of the x-axis (`0, -log10(1/length(pval))`). If the y-axis is truncated, then points off the top of the plot are denoted by triangles at the upper edge. The 95% confidence interval is shaded in gray.

If truncate is set to a numeric value, then `ylim` is set to `c(0, truncate)` only if the value of truncate is bigger than the maximum `-log10(pval)`. (Use the `ylim` argument if alternatve behavior is desired.)

If requested with `thinThreshold`, points with p-values `< -log10(thinThreshold)` are thinned before plotting. All points with `-log10(pval) >= thinThreshold` plus 10,000 points with `-log10(pval) < thinThreshold` (randomly selected in uniformly-spaced bins of -log10(pval)) are displayed.

Author(s)

Cathy Laurie, Matthew P. Conomos, Adrienne Stilp

Examples

```
pvals <- seq(0, 1, 0.001)
qqPlot(pvals)
qqPlot(pvals, thinThreshold=2)
qqPlot(pvals, truncate=TRUE)
qqPlot(pvals, truncate=10)
```

---

qualityScoreByScan *Mean and median quality score for scans*

Description

This function calculates the mean and median quality score, over all SNPs with a non-missing genotype call, for each scan.

Usage

```
qualityScoreByScan(intenData, genoData,
                   snp.exclude = NULL,
                   verbose = TRUE)
```

Arguments

`intenData`	`IntensityData` object
`genoData`	`GenotypeData` object
`snp.exclude`	An integer vector containing the id's of SNPs to be excluded.
`verbose`	Logical value specifying whether to show progress information.

Details

intenData and genoData must have matching snpID and scanID. Y chromosome SNPs are excluded for females. A "sex" variable must be present in the scan annotation slot of intenData or genoData.

Value

The function returns a matrix with the following columns:

mean.quality	A vector of mean quality scores for each scan
median.quality	A vector of median quality scores for each scan.

Author(s)

Cathy Laurie

See Also

IntensityData, GenotypeData, qualityScoreBySnp

Examples

```
library(GWASdata)
qualfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
qual <- GdsIntensityReader(qualfile)
# need scan annotation with sex
data(illuminaScanADF)
qualData <- IntensityData(qual, scanAnnot=illuminaScanADF)

genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <- GenotypeData(geno, scanAnnot=illuminaScanADF)

quality <- qualityScoreByScan(qualData, genoData)
close(qualData)
close(genoData)
```

---

qualityScoreBySnp *Mean and median quality score for SNPs*

Description

This function calculates the mean and median quality score, over all scans with a non-missing genotype call, for each SNP.

Usage

```
qualityScoreBySnp(intenData, genoData, scan.exclude = NULL,
                  block.size = 5000, verbose = TRUE)
```

Arguments

intenData	IntensityData object
genoData	GenotypeData object
scan.exclude	An integer vector containing the id's of scans to be excluded.
block.size	Number of SNPs to be read from intenData and genoData at once.
verbose	Logical value specifying whether to show progress information.

Details

intenData and genoData must have matching snpID and scanID.

Value

The function returns a matrix with the following columns:

mean.quality	A vector of mean quality scores for each snp.
median.quality	A vector of median quality scores for each snp.

Author(s)

Cathy Laurie

See Also

IntensityData, GenotypeData, qualityScoreByScan

Examples

```
qualfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
qual <- GdsIntensityReader(qualfile)
qualData <- IntensityData(qual)

genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
geno <- GdsGenotypeReader(genofile)
genoData <- GenotypeData(geno)

quality <- qualityScoreBySnp(qualData, genoData)
close(qualData)
close(genoData)
```

readWriteFirst *Read and write the first n lines of a file*

Description

Read first n lines of filein and write them to fileout, where filein and fileout are file names.

Usage

```
readWriteFirst(filein, fileout, n)
```

Arguments

filein	input file
fileout	output file
n	number of lines to write

Author(s)

Cathy Laurie

Examples

```
path <- system.file("extdata", "affy_raw_data", package="GWASdata")
file <- paste(path, list.files(path)[1], sep="/")
outf <- tempfile()
readWriteFirst(file, outf, 20)
file.remove(outf)
```

relationsMeanVar *Mean and Variance information for full-sibs, half-sibs, first-cousins*

Description

Computes theoretical mean and covariance matrix for k0 vs. k1 ibd coefficients for full-sib relationship along with inverse and eigenvalues/vectors of the covariance matrix.

Computes theoretical means and variances for half-sib relationship and for first-cousin relationship.

Usage

```
relationsMeanVar
```

Format

A list with the following entries:

`FullSibs` list with following entries:

- `mean`: mean of (k0,k1) for full-sibs
- `cov`: covariance matrix for full-sibs
- `invCov`: inverse of the covariance matrix
- `eigvals`: eigenvalues of the inverse covariance matrix
- `eigvectors`: eigenvectors of the inverse covariance matrix

`HalfSibs` list with following entries:

- `mean`: mean of (k0,k1) for half-sibs
- `var`: variance for half-sibs

`FirstCousins` list with following entries:

- `mean`: mean of (k0,k1) for first-cousins
- `var`: variance for first-cousin

Source

computed by Cecelia Laurie using the referenced papers

References

Hill, W.G. and B.S. Weir (2011) Variation in actual relationship as a consequence of Mendelian sampling and linkage,*Genet. Res., Camb.*, **93**, 47–64.

Kong, X., *et al* (2004) A combined physical-linkage map of the human genome, *American Journal of Human Genetics*, **75**, 1143–1148.

Examples

```
data(relationsMeanVar)
FS<-relationsMeanVar$FullSibs
FScov<-FS$cov   #gives covariance matrix for full-sibs
HS<-relationsMeanVar$HalfSibs
HSvar<-HS$var  #gives variance for half-sibs
```

---

saveas *Save an R object with a new name*

Description

Saves an R object as name in an Rdata file called path/name.RData.

Usage

```
saveas(obj, name, path=".")
```

Arguments

obj	R object to save
name	character string with the new name for the R object
path	path for the Rdata file (saved file will be path/name.RData)

Details

The suffix ".RData" will be appended to the new object name to create the file name, and the file will be written to the path directory.

Author(s)

Stephanie Gogarten

See Also

getobj

Examples

```
x <- 1:10
path <- tempdir()
saveas(x, "myx", path)
newfile <- paste(path, "/myx", ".RData", sep="")
load(newfile) # myx now loaded
unlink(newfile)
```

ScanAnnotationDataFrame

Class ScanAnotationDataFrame

Description

The ScanAnnotationDataFrame class stores annotation data associated with subjects in a genotyping study, where there may be multiple scans per subject, as well as metadata describing each column. It extends the AnnotatedDataFrame class.

Extends

AnnotatedDataFrame

Constructor

`ScanAnnotationDataFrame(data, metadata)`:

`data` must be a data.frame containing the scan annotation. It must contain at least the following column:

- "scanID": vector containing unique scan ids.

If a column representing sex is present, it must have the following format:

- "sex": character vector with values 'M' or 'F'.

`metadata` is an optional data.frame containing a description for each column in `data`. It should contain a column "labelDescription", with `row.names(metadata) == names(data)`.

The `ScanAnnotationDataFrame` constructor creates and returns a ScanAnnotationDataFrame instance.

Accessors

In the code snippets below, `object` is a ScanAnnotationDataFrame object.

`getScanID(object, index)`: A unique vector of scan IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getSex(object, index)`: A character vector of sex, with values 'M' or 'F'. The optional `index` is a logical or integer vector specifying elements to extract.

`hasSex(object)`: Returns `TRUE` if the column 'sex' is present in `object`.

`getVariable(object, varname, index)`: A vector of the column `varname`. The optional `index` is a logical or integer vector specifying elements to extract. If `varname` is itself a vector, returns a data.frame. Returns `NULL` if `varname` is not found in `object`.

`hasVariable(object, varname)`: Returns `TRUE` if `varname` is a column in `object`, `FALSE` if not.

`getVariableNames(object)`: Returns a character vector with the names of all columns in `object`.

`getAnnotation(object)`: Returns all annotation variables as a data frame.

`getMetadata(object)`: Returns metadata describing the annotation variables as a data frame.

Inherited methods from `AnnotatedDataFrame`:

`varLabels(object)`: Returns a character vector with the names of all columns in `object`.

`pData(object)`: Returns all annotation variables as a data frame, or sets the annotation variables with `pData(object) <- df`.

`varMetadata(object)`: Returns metadata describing the annotation variables as a data frame, or sets the metadata with `varMetadata(object) <- df`.

The operators `$` and `[` work just as they do in standard data frames, for both retrieval and assignment.

Author(s)

Stephanie Gogarten

See Also

`AnnotatedDataFrame`, `SnpAnnotationDataFrame`, `GenotypeData`, `IntensityData`

Examples

```
library(GWASdata)
data(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)

scanID <- getScanID(scanAnnot)
sex <- getSex(scanAnnot)
if (hasVariable(scanAnnot, "plate")) plate <- getVariable(scanAnnot, "plate")
subjectID <- getVariable(scanAnnot, "subjectID", index=(sex == "M"))

# list columns
varLabels(scanAnnot)

# add metadata
meta <- varMetadata(scanAnnot)
meta["scanID", "labelDescription"] <- "unique scan ID"
varMetadata(scanAnnot) <- meta

# display data
head(pData(scanAnnot))

# standard operators
scanID <- scanAnnot$scanID
sex <- scanAnnot[["sex"]]
subset <- scanAnnot[1:10, 1:5]
scanAnnot$newVar <- rep(1, nrow(scanAnnot))

# replace data
df <- pData(scanAnnot)
pData(scanAnnot) <- df
```

ScanAnnotationSQLite *Class ScanAnotationSQLite*

Description

The ScanAnnotationSQLite class stores annotation data associated with scans, as well as metadata describing each column, in an SQLite database.

Constructor

`ScanAnnotationSQLite(dbpath)`:

`dbpath` is the path to a SQLite database with tables "Annotation" and "Metadata." "Annotation" must contain at least the following column:

- "scanID": vector containing unique scan ids.

If a column representing sex is present, it must have the following format:

- "sex": character vector with values 'M' or 'F'.

"Metadata" must contain at least the following columns:

- "varname": name of variable in annotation
- "description": description of column in annotation

If the database does not yet exist, a database is created with tables "Annotation" and "Metadata."

The `ScanAnnotationSQLite` constructor creates and returns a ScanAnnotationSQLite instance.

Accessors

In the code snippets below, `object` is a ScanAnnotationSQLite object.

`open(object)`: Opens a connection to the database.

`close(object)`: Closes the database connection.

`nscan(object)`: The number of scans in the database.

`getScanID(object, index, condition)`: A unique vector of scan IDs. The optional `index` is a logical or integer vector specifying elements to extract. The optional `condition` is a character string with an SQL clause used to select data (e.g., "LIMIT 10", "WHERE sex='M'").

`getSex(object, index, condition)`: A character vector of sex, with values 'M' or 'F'. The optional `index` is a logical or integer vector specifying elements to extract. The optional `condition` is a character string with an SQL clause used to select data.

`hasSex(object)`: Returns `TRUE` if the column 'sex' is present in `object`.

`getVariable(object, varname, index, condition)`: A vector of the column `varname`. The optional `index` is a logical or integer vector specifying elements to extract. The optional `condition` is a character string with an SQL clause used to select data (e.g., "LIMIT 10", "WHERE sex='M'"). Returns `NULL` if `varname` is not found in `object`.

`hasVariable(object, varname)`: Returns `TRUE` if `varname` is a column in `object`, `FALSE` if not.

`getVariableNames(object)`: Returns a character vector with the names of all columns in `object`.

`getAnnotation(object)`: Returns all annotation variables as a data frame.

`getMetadata(object)`: Returns metadata describing the annotation variables as a data frame.

`getQuery(object, statement)`: Returns result of the SQL query `statement`.

`writeAnnotation(object, value, append=FALSE,overwrite=TRUE)`: Writes `value` to the scan annotation table. `value` must be a data.frame containing a column "scanID".

`writeMetadata(object, value, append=FALSE,overwrite=TRUE)`: Writes `value` to the metadata table. `value` should be a data.frame containing columns "varname" and "description".

Author(s)

Stephanie Gogarten

See Also

SnpAnnotationSQLite, ScanAnnotationDataFrame, GenotypeData, IntensityData

Examples

```
library(GWASdata)
dbpath <- tempfile()
scanAnnot <- ScanAnnotationSQLite(dbpath)

data(illumina_scan_annot)
writeAnnotation(scanAnnot, illumina_scan_annot)

# list columns
vars <- getVariableNames(scanAnnot)

# add metadata
metadf <- data.frame(varname=vars, description=rep(NA, length(vars)),
  row.names=vars, stringsAsFactors=FALSE)
metadf["scanID", "description"] <- "unique id"
writeMetadata(scanAnnot, metadf)

scanID <- getScanID(scanAnnot)
sex <- getSex(scanAnnot)
if (hasVariable(scanAnnot, "plate")) plate <- getVariable(scanAnnot, "plate")
subjectID <- getVariable(scanAnnot, "subjectID", condition="WHERE sex=M")

# display data
head(getAnnotation(scanAnnot))
getMetadata(scanAnnot)

close(scanAnnot)
file.remove(dbpath)
```

---

setMissingGenotypes *Write a new netCDF or GDS file, setting certain SNPs to missing*

Description

setMissingGenotypes copies an existing GDS or netCDF genotype file to a new one, setting SNPs in specified regions to missing.

Usage

```
setMissingGenotypes(parent.file, new.file, regions, file.type=c("gds", "ncdf"),
                    sample.include=NULL, compress="ZIP.max", verbose=TRUE)
```

Arguments

parent.file	Name of the parent file
new.file	Name of the new file
regions	Data.frame of chromosome regions with columns "scanID", "chromosome", "left.base", "right.ba
file.type	The type of parent.file and new.file ("gds" or "ncdf")

sample.include	Vector of sampleIDs to include in new.file
compress	the compression format for the GDS file, one of "", "ZIP", "ZIP.fast", "ZIP.default", or "ZIP.max"
verbose	Logical value specifying whether to show progress information.

Details

setMissingGenotypes removes chromosome regions by setting SNPs that fall within the anomaly regions to NA (i.e., the missing value in the netCDF/GDS file). Optionally, entire samples may be excluded from the netCDF/GDS file as well: if the sample.include argument is given, only the scanIDs in this vector will be written to the new file, so the sample dimension will be length(sample.include).

For regions with whole.chrom=TRUE, the entire chromosome will be set to NA for that sample. For other regions, only the region between left.base and right.base will be set to NA.

Author(s)

Stephanie Gogarten

See Also

ncdfSubset, gdsSubset, anomSegStats for chromosome anomaly regions

Examples

```
gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
gds <- GdsGenotypeReader(gdsfile)
sample.sel <- getScanID(gds, index=1:10)
close(gds)

regions <- data.frame("scanID"=sample.sel[1:3], "chromosome"=c(21,22,23),
  "left.base"=c(14000000, 30000000, NA), "right.base"=c(28000000, 450000000, NA),
  whole.chrom=c(FALSE, FALSE, TRUE))

newgds <- tempfile()
setMissingGenotypes(gdsfile, newgds, regions, file.type="gds", sample.include=sample.sel)
file.remove(newgds)
```

---

simulateGenotypeMatrix

Simulate Genotype Matrix & Load into NetCDF File

Description

This function creates a netCDF file with dimensions 'snp' and 'sample' and variables 'sampleID', 'genotype', 'position' and 'chromosome'. These variables hold simulated data as described below. Mainly, this function is intended to be used in examples involving genotype matrices.

Usage

```
simulateGenotypeMatrix(n.snps=10, n.chromosomes=10,
                       n.samples=1000, ncdf.filename,
                       silent=TRUE)
```

Arguments

n.snps	An integer corresponding to the number of SNPs per chromosome, the default value is 10. For this function, the number of SNPs is assumed to be the same for every chromosome.
n.chromosomes	An integer value describing the total number of chromosomes with default value 10.
n.samples	An integer representing the number of samples for our data. The default value is 1000 samples.
ncdf.filename	A string that will be used as the name of the netCDF file. This is to be used later when opening and retrieving data generated from this function.
silent	Logical value. If FALSE, the function returns a table of genotype counts generated. The default is TRUE; no data will be returned in this case.

Details

The resulting netCDF file will have the following characteristics:

Dimensions:

'snp': n.snps*n.chromosomes length

'sample': n.samples length

Variables:

'sampleID': sample dimension, values 1-n.samples

'position': snp dimension, values [1,2,...,n.chromosomes] n.snps times

'chromosome': snp dimension, values [1,1,...]n.snps times, [2,2,...]n.snps times, ..., [n.chromosomes,n.chromosomes,...]n.snp
times

'genotype': 2-dimensional snp x sample, values 0, 1, 2 chosen from allele frequencies that were generated from a uniform distribution on (0,1). The missing rate is 0.05 (constant across all SNPs) and is denoted by -1.

Value

This function returns a table of genotype calls if the silent variable is set to FALSE, where 2 indicates an AA genotype, 1 is AB, 0 is BB and -1 corresponds to a missing genotype call.

A netCDF file is created from this function and written to disk. This file (and data) can be accessed later by using the command `open.ncdf(ncdf.filename)`.

Author(s)

Caitlin McHugh

See Also

ncdf, missingGenotypeBySnpSex, missingGenotypeByScanChrom, simulateIntensityMatrix

Examples

```
filenm <- tempfile()

simulateGenotypeMatrix(ncdf.filename=filenm )

file <- NcdfGenotypeReader(filenm)
file #notice the dimensions and variables listed

genot <- getGenotype(file)
table(genot) #can see the number of missing calls

chrom <- getChromosome(file)
unique(chrom) #there are indeed 10 chromosomes, as specified in the function call

close(file)
unlink(filenm)
```

---

simulateIntensityMatrix

Simulate Intensity Matrix & Load into NetCDF File

Description

This function creates a netCDF file with dimensions 'snp' and 'sample' and variables 'sampleID', 'position', 'chromosome', 'quality', 'X', and 'Y'. These variables hold simulated data as explained below. Mainly, this function is intended to be used in examples involving matrices holding quantitative data.

Usage

```
simulateIntensityMatrix(n.snps=10, n.chromosomes=10,
                        n.samples=1000, ncdf.filename,
                        silent=TRUE)
```

Arguments

n.snps	An integer corresponding to the number of SNPs per chromosome, the default value is 10. For this function, the number of SNPs is assumed to be the same for every chromosome.
n.chromosomes	An integer value describing the total number of chromosomes with default value 10.
n.samples	An integer representing the number of samples for our data. The default value is 1000 samples.

`ncdf.filename`	A string that will be used as the name of the netCDF file. This is to be used later when opening and retrieving data generated from this function.
`silent`	Logical value. If `FALSE`, the function returns a list of heterozygosity and missing values. The default is `TRUE`; no data will be returned in this case.

Details

The resulting netCDF file will have the following characteristics:

Dimensions:

'snp': n.snps*n.chromosomes length

'sample': n.samples length

Variables:

'sampleID': sample dimension, values 1-n.samples

'position': snp dimension, values [1,2,...,n.chromosomes] n.snps times

'chromosome': snp dimension, values[1,1,...]n.snps times, [2,2,...]n.snps times, ... , [n.chromosomes,n.chromosomes,...]n.snp times

'quality': 2-dimensional snp x sample, values between 0 and 1 chosen randomly from a uniform distribution. There is one quality value per snp, so this value is constant across all samples.

'X': 2-dimensional snp x sample, value of X intensity taken from a normal distribution. The mean of the distribution for each SNP is based upon the sample genotype. Mean is 0,2 if sample is homozygous, 1 if heterozygous.

'Y': 2-dimensional snp x sample, value of Y intensity also chosen from a normal distribution, where the mean is chosen according to the mean of X so that sum of means = 2.

Value

This function returns a list if the silent variable is set to FALSE, which includes:

`het`	Heterozygosity table
`nmiss`	Number of missing values

A netCDF file is created from this function and written to disk. This file (and data) can be accessed later by using the command 'open.ncdf(ncdf.filename)'.

Author(s)

Caitlin McHugh

See Also

ncdf, `meanIntensityByScanChrom`, `simulateGenotypeMatrix`

Examples

```
filenm <- tempfile()

simulateIntensityMatrix(ncdf.filename=filenm, silent=FALSE )

file <- NcdfIntensityReader(filenm)
file #notice the dimensions and variables listed

xint <- getX(file)
yint <- getY(file)
print("Number missing is: "); sum(is.na(xint))

chrom <- getChromosome(file)
unique(chrom) #there are indeed 10 chromosomes, as specified in the function call

close(file)
unlink(filenm)
```

SnpAnnotationDataFrame

Class SnpAnotationDataFrame

Description

The SnpAnnotationDataFrame class stores annotation data associated with SNPs, as well as metadata describing each column. It extends the AnnotatedDataFrame class.

Extends

AnnotatedDataFrame

Constructor

SnpAnnotationDataFrame(data, metadata):

data must be a data.frame containing the SNP annotation. It must contain at least the following columns:

- "snpID": integer vector containing unique SNP ids.
- "chromosome": integer vector containing chromosome codes.
- "position": integer vector containing position (in base pairs) on the chromosome.

Default values for chromosome codes are 1-22=autosome, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments autosomeCode, XchromCode, XYchromCode, YchromCode, and MchromCode.

metadata is an optional data.frame containing a description for each column in data. It should contain a column "labelDescription", with row.names(metadata) == names(data).

The SnpAnnotationDataFrame constructor creates and returns a SnpAnnotationDataFrame instance.

Accessors

In the code snippets below, `object` is a SnpAnnotationDataFrame object.

`getSnpID(object, index)`: A unique integer vector of snp IDs. The optional `index` is a logical or integer vector specifying elements to extract.

`getChromosome(object, index, char=FALSE)`: A vector of chromosomes. The optional `index` is a logical or integer vector specifying elements to extract. If `char=FALSE` (default), returns an integer vector. If `char=TRUE`, returns a character vector with elements in (1:22,X,XY,Y,M,U). "U" stands for "Unknown" and is the value given to any chromosome code not falling in the other categories.

`getPosition(object, index)`: An integer vector of base pair positions. The optional `index` is a logical or integer vector specifying elements to extract.

`getAlleleA(object, index)`: A character vector of A alleles. The optional `index` is a logical or integer vector specifying elements to extract.

`getAlleleB(object, index)`: A character vector of B alleles. The optional `index` is a logical or integer vector specifying elements to extract.

`getVariable(object, varname, index)`: A vector of the column `varname`. The optional `index` is a logical or integer vector specifying elements to extract. If `varname` is itself a vector, returns a data.frame. Returns `NULL` if `varname` is not found in `object`.

`hasVariable(object, varname)`: Returns `TRUE` if `varname` is a column in `object`, `FALSE` if not.

`getVariableNames(object)`: Returns a character vector with the names of all columns in `object`.

`getAnnotation(object)`: Returns all annotation variables as a data frame.

`getMetadata(object)`: Returns metadata describing the annotation variables as a data frame.

Inherited methods from AnnotatedDataFrame:

`varLabels(object)`: Returns a character vector with the names of all columns in `object`.

`pData(object)`: Returns all annotation variables as a data frame, or sets the annotation variables with `pData(object) <- df`.

`varMetadata(object)`: Returns metadata describing the annotation variables as a data frame, or sets the metadata with `varMetadata(object) <- df`.

The operators `[`, `$`, and `[[` work just as they do in standard data frames, for both retrieval and assignment.

`autosomeCode(object)`: Returns the integer codes for the autosomes.

`XchromCode(object)`: Returns the integer code for the X chromosome.

`XYchromCode(object)`: Returns the integer code for the pseudoautosomal region.

`YchromCode(object)`: Returns the integer code for the Y chromosome.

`MchromCode(object)`: Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

AnnotatedDataFrame, ScanAnnotationDataFrame, GenotypeData, IntensityData

Examples

```
library(GWASdata)
data(illumina_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(illumina_snp_annot)

# list columns
varLabels(snpAnnot)

# add metadata
meta <- varMetadata(snpAnnot)
meta["snpID", "labelDescription"] <- "unique integer ID"
varMetadata(snpAnnot) <- meta

# get snpID and chromosome
snpID <- getSnpID(snpAnnot)
chrom <- getChromosome(snpAnnot)

# get positions only for chromosome 22
pos22 <- getPosition(snpAnnot, index=(chrom == 22))

# get rsID
if (hasVariable(snpAnnot, "rsID")) rsID <- getVariable(snpAnnot, "rsID")

# display data
head(pData(snpAnnot))

# standard operators
snpID <- snpAnnot$snpID
chrom <- snpAnnot[["chromosome"]]
subset <- snpAnnot[1:10, 1:5]
snpAnnot$newVar <- rep(1, nrow(snpAnnot))

# replace data
df <- pData(snpAnnot)
pData(snpAnnot) <- df

# PLINK chromosome coding
snpID <- 1:10
chrom <- c(rep(1L,5), 23:27)
pos <- 101:110
df <- data.frame(snpID=snpID, chromosome=chrom, position=pos)
snpAnnot <- SnpAnnotationDataFrame(df, YchromCode=24L, XYchromCode=25L)
getChromosome(snpAnnot, char=TRUE)
```

SnpAnnotationSQLite *Class SnpAnotationSQLite*

Description

The SnpAnnotationSQLite class stores annotation data associated with SNPs, as well as metadata describing each column, in an SQLite database.

Constructor

`SnpAnnotationSQLite(dbpath)`:

`dbpath` is the path to a SQLite database with tables "Annotation" and "Metadata." "Annotation" must contain at least the following columns:

- "snpID": integer vector containing unique SNP ids.
- "chromosome": integer vector containing chromosome codes.
- "position": integer vector containing position (in base pairs) on the chromosome.

Default values for chromosome codes are 1-22=autosome, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `autosomeCode`, `XchromCode`, `XYchromCode`, `YchromCode`, and `MchromCode`.

"Metadata" must contain at least the following columns:

- "varname": name of variable in annotation
- "description": description of column in annotation

If the database does not yet exist, a database is created with tables "Annotation" and "Metadata."

The `SnpAnnotationSQLite` constructor creates and returns a SnpAnnotationSQLite instance.

Accessors

In the code snippets below, `object` is a SnpAnnotationSQLite object.

`open(object)`: Opens a connection to the database.

`close(object)`: Closes the database connection.

`nsnp(object)`: The number of SNPs in the database.

`getSnpID(object, index, condition)`: A unique integer vector of snp IDs. The optional `index` is a logical or integer vector specifying elements to extract. The optional `condition` is a character string with an SQL clause used to select data (e.g., "LIMIT 10", "WHERE chromosome=1").

`getChromosome(object, index, condition, char=FALSE)`: A vector of chromosomes. The optional `index` is a logical or integer vector specifying elements to extract. The optional `condition` is a character string with an SQL clause used to select data (e.g., "LIMIT 10", "WHERE chromosome=1"). If `char=FALSE` (default), returns an integer vector. If `char=TRUE`, returns a character vector with elements in (1:22,X,XY,Y,M,U). "U" stands for "Unknown" and is the value given to any chromosome code not falling in the other categories.

`getPosition(object, index, condition)`: An integer vector of base pair positions. The optional `index` is a logical or integer vector specifying elements to extract. The optional `condition` is a character string with an SQL clause used to select data (e.g., "LIMIT 10", "WHERE chromosome=1").

`getAlleleA(object, index)`: A character vector of A alleles. The optional `condition` is a character string with an SQL clause used to select data (e.g., "LIMIT 10", "WHERE chromosome=1").

`getAlleleB(object, index)`: A character vector of B alleles. The optional `condition` is a character string with an SQL clause used to select data (e.g., "LIMIT 10", "WHERE chromosome=1").

`getVariable(object, varname, index, condition)`: A vector of the column `varname`. The optional `index` is a logical or integer vector specifying elements to extract. The optional `condition` is a character string with an SQL clause used to select data (e.g., "LIMIT 10", "WHERE chromosome=1"). Returns `NULL` if `varname` is not found in `object`.

`hasVariable(object, varname)`: Returns `TRUE` if `varname` is a column in `object`, `FALSE` if not.

`getVariableNames(object)`: Returns a character vector with the names of all columns in `object`.

`getAnnotation(object)`: Returns all annotation variables as a data frame.

`getMetadata(object)`: Returns metadata describing the annotation variables as a data frame.

`getQuery(object, statement)`: Returns result of the SQL query `statement`.

`writeAnnotation(object, value, append=FALSE,overwrite=TRUE)`: Writes `value` to the SNP annotation table. `value` must be a data.frame containing columns "snpID", "chromosome", and "position".

`writeMetadata(object, value, append=FALSE,overwrite=TRUE)`: Writes `value` to the metadata table. `value` should be a data.frame containing columns "varname" and "description".

`autosomeCode(object)`: Returns the integer codes for the autosomes.

`XchromCode(object)`: Returns the integer code for the X chromosome.

`XYchromCode(object)`: Returns the integer code for the pseudoautosomal region.

`YchromCode(object)`: Returns the integer code for the Y chromosome.

`MchromCode(object)`: Returns the integer code for mitochondrial SNPs.

Author(s)

Stephanie Gogarten

See Also

ScanAnnotationSQLite, SnpAnnotationDataFrame, GenotypeData, IntensityData

Examples

```
library(GWASdata)
dbpath <- tempfile()
snpAnnot <- SnpAnnotationSQLite(dbpath)

data(illumina_snp_annot)
writeAnnotation(snpAnnot, illumina_snp_annot)

# list columns
vars <- getVariableNames(snpAnnot)

# add metadata
metadf <- data.frame(varname=vars, description=rep(NA, length(vars)),
  row.names=vars, stringsAsFactors=FALSE)
metadf["snpID", "description"] <- "integer id"
writeMetadata(snpAnnot, metadf)

# get snpID and chromosome
```

```
snpID <- getSnpID(snpAnnot)
chrom <- getChromosome(snpAnnot)

# get positions only for chromosome 22
pos22 <- getPosition(snpAnnot, condition="WHERE chromosome = 22")

# get rsID
if (hasVariable(snpAnnot, "rsID")) rsID <- getVariable(snpAnnot, "rsID")

# display data
head(getAnnotation(snpAnnot))
getMetadata(snpAnnot)

close(snpAnnot)
file.remove(dbpath)
```

`snpCorrelationPlot` *SNP correlation plot*

Description

Plots SNP correlation versus chromosome.

Usage

```
snpCorrelationPlot(correlations, chromosome,
                   ylim=c(0,1), ylab = "abs(correlation)", ...)
```

Arguments

`correlations`	A vector of correlations.
`chromosome`	A vector containing the chromosome for each SNP.
`ylim`	The limits of the y axis.
`ylab`	The label for the y axis.
`...`	Other parameters to be passed directly to `plot`.

Details

Plots SNP correlations (from, e.g., PCA), versus chromosome.

`correlations` must have the same length as `chromosome` and is assumed to be in order of position on each chromosome. Values within each chromosome are evenly spaced along the X axis.

Author(s)

Cathy Laurie

See Also

manhattanPlot

Examples

```
correlations <- sample(0.001*(0:1000), 1000, replace=TRUE)
chromosome <- c(rep(1,400), rep(2,350), rep("X",200), rep("Y",50))
snpCorrelationPlot(correlations, chromosome)
```

---

vcfWrite *Utility to write VCF file*

Description

vcfWrite creates a VCF file from a GenotypeData object.

Usage

```
vcfWrite(genoData, vcf.file="out.vcf", sample.col="scanID",
         id.col="snpID", qual.col=NULL, filter.cols=NULL,
         info.cols=NULL, scan.exclude=NULL, snp.exclude=NULL,
         ref.allele=NULL, block.size=1000, verbose=TRUE)

vcfCheck(genoData, vcf.file, sample.col="scanID",
         id.col="snpID", block.size=1000, verbose=TRUE)
```

Arguments

genoData	A GenotypeData object with scan and SNP annotation.
vcf.file	Filename for the output VCF file.
sample.col	name of the column in the scan annotation to use as sample IDs in the VCF file
id.col	name of the column in the SNP annotation to use as "ID" column in the VCF file
qual.col	name of the column in the SNP annotation to use as "QUAL" column in the VCF file
filter.cols	vector of column names in the SNP annotation to use as "FILTER" column in the VCF file. These columns should be logical vectors, with TRUE for SNPs to be filtered. Any SNPs with a value of FALSE for all filter columns will be set to "PASS".
info.cols	vector of column names in the SNP annotation to concatenate for the "INFO" column in the VCF file.
scan.exclude	vector of scanIDs to exclude from VCF file
snp.exclude	vector of snpIDs to exclude from VCF file
ref.allele	vector of "A" or "B" values indicating where allele A or allele B should be the reference allele for each SNP. Default is to use allele A as the reference allele.

block.size	Number of SNPs to read from genoData at a time
verbose	logical for whether to show progress information.

Details

REF will be alleleA and ALT will be alleleB.

vcfCheck compares the genotypes (diploid only) in a VCF file to the corresponding genotypes in genoData. It stops with an error when it detects a discordant genotype. It assumes that the "ID" column of the VCF file has unique values that can be matched with a column in the SNP annotation, and that all SNPs in the VCF file are present in genoData.

Author(s)

Stephanie Gogarten

References

The variant call format and VCFtools. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. Bioinformatics. 2011 Aug 1;27(15):2156-8. Epub 2011 Jun 7.

See Also

snpgdsVCF2GDS

Examples

```
library(GWASdata)
library(VariantAnnotation)
gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
data(illuminaSnpADF, illuminaScanADF)
genoData <- GenotypeData(GdsGenotypeReader(gdsfile),
  scanAnnot=illuminaScanADF, snpAnnot=illuminaSnpADF)
vcffile <- tempfile()
vcfWrite(genoData, vcffile, id.col="rsID", info.cols="IntensityOnly")
vcf <- readVcf(vcffile, "hg18")
vcf
vcfCheck(genoData, vcffile, id.col="rsID")
close(genoData)
unlink(vcffile)
```

Index