GWASTools

March 24, 2012

`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. The values are written to a netCDF file which is assumed to exist with proper variables and size.

Usage

```
BAFfromClusterMeans(intenData, bl.ncdf.filename,
                    clusterMeanVars = c("tAA","tAB","tBB","rAA","rAB","rBB"),
                    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.
`bl.ncdf.filename`	The filepath for a previously created netCDF file to hold the B allele frequency and log R ratio values.
`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).
`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`. The file specified by `bl.ncdf.filename` is assumed to have variables 'BAlleleFreq' and 'LogRRatio' to which the proper values are written.

Value

The netCDF file stored in the `bl.ncdf.filename` path is populated with values of B allele frequency and the log R ratio at the completion of this function.

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 netCDF
library(GWASdata)
xyfile <- system.file("extdata", "illumina_qxy.nc", package="GWASdata")
xyNC <- NcdfIntensityReader(xyfile)
data(illumina_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(illumina_snp_annot)
xyData <- IntensityData(xyNC, snpAnnot=snpAnnot)
nsamp <- nscan(xyData)

# create netCDF file to hold BAF/LRR data
blfile <- tempfile()
ncdfCreate(illumina_snp_annot, blfile, variables=c("BAlleleFreq","LogRRatio"), n.samples=

# calculate BAF and LRR
BAFfromClusterMeans(xyData, blfile, verbose=FALSE)

# read output
blNC <- NcdfIntensityReader(blfile)
blData <- IntensityData(blNC)
baf <- getBAlleleFreq(blData)
lrr <- getLogRRatio(blData)

close(xyNC)
close(blNC)
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. The values are written to a netCDF file which is assumed to exist with proper variables and size.

Usage

```
BAFfromGenotypes(intenData, genoData,
                 bl.ncdf.filename, min.n.genotypes = 2,
                 call.method = c("by.plate", "by.study"),
                 plate.name = "plate",
                 block.size = 5000, 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.
bl.ncdf.filename	The filepath for a previously created netCDF file to hold the B allele frequency and log R ratio values.
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 from the netCDF file at one time. The recommended value is around 1000, but should vary depending on computing power.
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. The file specified by bl.ncdf.filename is assumed to have variables 'BAlleleFreq' and 'LogRRatio' to which the proper values are written.

Value

The netCDF file stored in the bl.ncdf.filename path is populated with values of B allele frequency and the log R ratio at the completion of this function.

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(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
nsamp <- nrow(scanAnnot)

data(affy_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(affy_snp_annot)

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

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

# create netCDF file to hold BAF/LRR data
blfile <- tempfile()
ncdfCreate(affy_snp_annot, blfile, variables=c("BAlleleFreq","LogRRatio"), n.samples=nsam

# calculate BAF and LRR
BAFfromGenotypes(xyData, genoData, blfile, 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)
```

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 NetCDF files, so it is possible to analyze data sets that are too large to be contained in memory. The `NcdfReader` class provides a generic interface to the NetCDF files (utilizing the `ncdf` package), and the `NcdfGenotypeReader` and `NcdfIntensityReader` classes provide specific methods to access genotype and intensity data.

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 Gogarten, Cathy Laurie, Tushar Bhangale, Matt Conomos, Cecilia Laurie, Caitlin McHugh, Ian Painter, Xiuwen Zheng, Rohit Swarnkar

Maintainer: Stephanie 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

`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` 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.

`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)`: 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`.

`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.

`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`, `NcdfReader`, `NcdfGenotypeReader`, `MatrixGenotypeReader`, `IntensityData`

Examples

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

# object without annotation
genoData <- GenotypeData(nc)

# object with annotation
data(affy_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(affy_snp_annot)
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
genoData <-  GenotypeData(nc, snpAnnot=snpAnnot, scanAnnot=scanAnnot)

# 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))

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)
```

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 an `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.

`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, NcdfReader, NcdfIntensityReader, GenotypeData

Examples

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

# object without annotation
intenData <- IntensityData(nc)

# object with annotation
data(affy_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(affy_snp_annot)
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
intenData <-  IntensityData(nc, snpAnnot=snpAnnot, scanAnnot=scanAnnot)

# 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)
```

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, scanID=scanID)`:

`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, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `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`.

`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

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))
```

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, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `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`.

`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", "affy_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)
```

NcdfIntensityReader

Class NcdfIntensityReader

Description

The NcdfIntensityReader class is an extension of the NcdfReader class specific to reading genotype 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, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `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.

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", "affy_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 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.

`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).

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

`close(object)`: Closes the NetCDF file connection.

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.

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)
```

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": integer 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 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`.

`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(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_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 integer 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": integer 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 integer 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(affy_scan_annot)
writeAnnotation(scanAnnot, affy_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"] <- "integer 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)
```

`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, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `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.

`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.

`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(affy_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(affy_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, 23=X, 24=XY, 25=Y, 26=M. The defaults may be changed with the arguments `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").

`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".

`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(affy_snp_annot)
writeAnnotation(snpAnnot, affy_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)
```

---

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 3 columns ("M" for males, "F" for females, "all" for all scans).

Author(s)

Cathy Laurie

See Also

GenotypeData

Examples

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

# need scan annotation with sex
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
genoData <-  GenotypeData(nc, scanAnnot=scanAnnot)

afreq <- alleleFrequency(genoData, scan.exclude=(scanAnnot$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 NAs 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)

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. Recommended to include all autosomes.
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 chromosome 23 (X). See HLA and pseudoautosomal.
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.
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
...	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.m 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 chromosome 23 (X) 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 where 23 refers to X chromosome
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(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
data(illumina_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(illumina_snp_annot)

blfile <- system.file("extdata", "illumina_bl.nc", package="GWASdata")
blnc <- NcdfIntensityReader(blfile)
blData <-  IntensityData(blnc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

genofile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
genonc <- NcdfGenotypeReader(genofile)
genoData <-  GenotypeData(genonc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

# segment BAF
scan.ids <- scanAnnot$scanID[1:2]
chrom.ids <- unique(snpAnnot$chromosome)
snp.ids <- snpAnnot$snpID[snpAnnot$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)
```

---

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
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 chromosome 23 (X). See HLA and pseudoautosomal.
known.anoms	data.frame of known anomalies (usually from anomDetectBAF); must have "scanID","chromosome","left.index","right.index". 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 chromosome 23 (X) 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 with 23 representing X

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 with 23 representing X
- 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, with 23 representing X
- 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, with 23 representing X
- 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, with 23 representing X

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(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
data(illumina_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(illumina_snp_annot)

blfile <- system.file("extdata", "illumina_bl.nc", package="GWASdata")
blnc <- NcdfIntensityReader(blfile)
```

```
blData <-  IntensityData(blnc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

genofile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
genonc <- NcdfGenotypeReader(genofile)
genoData <-  GenotypeData(genonc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

scan.ids <- scanAnnot$scanID[1:2]
chrom.ids <- unique(snpAnnot$chromosome)
snp.ids <- snpAnnot$snpID[snpAnnot$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)
```

`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(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
data(illumina_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(illumina_snp_annot)

blfile <- system.file("extdata", "illumina_bl.nc", package="GWASdata")
blnc <- NcdfIntensityReader(blfile)
blData <-  IntensityData(blnc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

genofile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
genonc <- NcdfGenotypeReader(genofile)
genoData <-  GenotypeData(genonc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

# 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 <- scanAnnot$scanID[1:2]
chrom.ids <- unique(snpAnnot$chromosome)
snp.ids <- snpAnnot$snpID[snpAnnot$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
snpAnnot$eligible <- snpAnnot$missing.n1 < 1
low.qual <- anomIdentifyLowQuality(snpAnnot, 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"), info = NULL, cex = 0.5)
```

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 chromosome 23 (X). See HLA and pseudoautosomal.
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.
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 statstics, 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
info	character vector of extra information to include in the main title of the upper plot
cex	cex value for points on the plots

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 stored in the working directory from which the program is called. 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(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
data(illumina_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(illumina_snp_annot)

blfile <- system.file("extdata", "illumina_bl.nc", package="GWASdata")
blnc <- NcdfIntensityReader(blfile)
blData <-  IntensityData(blnc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

genofile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
genonc <- NcdfGenotypeReader(genofile)
genoData <-  GenotypeData(genonc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)

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

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

# 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=2)

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)
```

`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)
```

---

assocTestCPH *Cox proportional hazards*

Description

Fits Cox proportional hazards model

Usage

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

Arguments

genoData	GenotypeData object, should contain sex and phenotypes in scan annotation
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)
stratum	name of variable 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:

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(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
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
ncfile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(ncfile)
genoData <-  GenotypeData(nc, 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)
```

`assocTestRegression` *Association tests*

Description

This function performs regression based association tests. It also performs genotype counts for association tests.

Usage

```
assocTestRegression(genoData, outcome, model.type,
covar.list = NULL, ivar.list = NULL,
                    gene.action.list = NULL,
                    scan.chromosome.filter = NULL,
                    scan.exclude = NULL, CI = 0.95,
                    robust = FALSE, geno.counts = 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
`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.

`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, and for the linear trend parameter when using linear 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.

`geno.counts` if `TRUE` (default), genotype counts are computed for each linear or logistic model. For linear models, counts are performed over all samples; for logistic models, counts are performed separately for cases and controls.

`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. When using a model with interaction terms, the association tests compare the model including all the interactions, the covariates, and the genotype value to the model with only the covariates and genotype value (jointly testing for all the interaction effects). All tests and p-values are found using Wald tests. 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.

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.

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.

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 three columns of each data-frame are:

snpID — snpID (from genoData) of the SNP

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.

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

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

model.N.gene_action.n — sample size for the regression

model.N.gene_action.warningOrError — warning or error occured during model fitting (1 if warning or error, NA if none)

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

model.N.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 with interaction variables: Here, "ivar_name" refers to the name of the interaction variable; if there are multiple interaction variables, there will be a column with each different "ivar_name".

model.N.gene_action.Est.G.ivar_name — estimate of the regression coefficient for the interaction between genotype and ivar_name.

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

For tests that use logistic regression with no interaction variables:

`model.N.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.N.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.N.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 logistic regression and interaction variables:

`model.N.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.N.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.N.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 linear or poisson regression with no interaction variables:

`model.N.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.N.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 linear or poisson regression and interaction variables:

`model.N.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.N.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 with no interaction variables:

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

`model.N.gene_action.pvalue.G`
: Wald test p-value. This can be calculated using either sandwich based robust standard errors or model based standard errors (see `robust`).

For tests with interaction variables:

`model.N.gene_action.Stat.GxE`
: value of the Wald test statistic for jointly testing all genotype interaction parameters

`model.N.gene_action.pvalue.GxE`
: Wald test p-value for jointly testing all genotype interaction parameters. This can be calculated using either sandwich based robust standard errors or model based standard errors (see `robust`).

If `geno.counts = TRUE`, for tests that use linear regression:

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

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

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

If `geno.counts = TRUE`, for tests that use logistic regression:

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

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

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

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

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

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

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 ~ 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)

Tushar Bhangale, Matt Conomos

See Also

`GenotypeData`, `lm`, `glm`, `vcov`, `vcovHC`

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, includin
# 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 u
# This test would only use chromosome 21.  It would also use sandwich based robust standa

# an example of a scan chromosome matrix
# desiged to eliminate duplicated individuals
# and scans with missing values of sex
library(GWASdata)
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_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$case.cntl.
scanAnnot$blood.pressure[scanAnnot$case.cntl.status==0] <- rnorm(sum(scanAnnot$case.cntl.

# create data object
ncfile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(ncfile)
genoData <-  GenotypeData(nc, 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,
                        geno.counts = 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 par
# 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
# these values are retrieved in R by putting a
# chr.set <- as.numeric(commandArgs(trailingOnly=TRUE))
# command in assoc.analysis.r
```

`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 = TRUE,
                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, each output matrix will have only one column.

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.

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", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(file)
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
genoData <-  GenotypeData(nc, scanAnnot=scanAnnot)

# 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")
qqPlot(res.fisher$pval)
```

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"),
                   code = NULL, main.txt = NULL,
                   abln = NULL,  horizln = c(1/2, 1/3, 2/3),
                   colorGenotypes = FALSE, genoData = NULL,
                   colorBatch = FALSE, batch.column = NULL,
                   snp.exclude = NULL, ...)
```

Arguments

intenData	IntensityData object, must contain at least one of 'BAlleleFreq', 'LogRRatio', 'X', 'Y'.
scan.ids	A vector containing the sample indices of the plots.
chrom.ids	A vector containing the chromosome indices of the plots.
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.
code	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.
main.txt	Text that will be written in the title on all plots created.
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.
...	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(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
blfile <- system.file("extdata", "illumina_bl.nc", package="GWASdata")
blnc <- NcdfIntensityReader(blfile)
intenData <-  IntensityData(blnc, scanAnnot=scanAnnot)

genofile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
genonc <- NcdfGenotypeReader(genofile)
genoData <-  GenotypeData(genonc, scanAnnot=scanAnnot)

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

---

convertNcdfGds *Convert between NetCDF and GDS format*

Description

`convertNcdfGds` converts a genotype NetCDF file to the GDS format used by the R packages `gdsfmt` and `SNPRelate`.

`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,
  sample.annot = NULL, snp.annot = NULL, rsID.col = "rsID",
  alleleA.col = "alleleA", alleleB.col = "alleleB",
  zipflag = "zip.max", verbose = TRUE)

convertGdsNcdf(gds.filename, ncdf.filename, verbose = TRUE)

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

Arguments

`ncdf.filename`	name of the NetCDF genotype file
`gds.filename`	name of the GDS file
`sample.annot`	a data.frame with sample annotation
`snp.annot`	a data.frame with SNP annotation
`rsID.col`	the name of the `snp.annot` column with rs ID
`alleleA.col`	the name of the `snp.annot` column with allele A
`alleleB.col`	the name of the `snp.annot` column with allele B

zipflag	the compression format for the GDS file, one of "", "ZIP", "ZIP.fast", "ZIP.default", or "ZIP.max"
verbose	whether to show progress information

Details

These functions require that the package gdsfmt be installed. convertNcdfGds is needed to convert the NetCDF genotype files used in this package to the format required by SNPRelate for Principal Component Analysis, Identity-by-Descent, and Linkage Disequlibrium calculations.

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

See Also

gdsfmt, SNPRelate, ncdf, NcdfGenotypeReader,

Examples

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

data(illumina_snp_annot)
data(illumina_scan_annot)

gdsfile <- tempfile()
convertNcdfGds(ncfile, gdsfile, sample.annot=illumina_scan_annot,
  snp.annot=illumina_snp_annot, rsID.col="rsID",
  alleleA.col="allele.A", alleleB.col="allele.B")

checkNcdfGds(ncfile, gdsfile)

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

file.remove(gdsfile, ncfile2)
```

---

duplicateDiscordance

Duplicate discordance

Description

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

Usage

```
duplicateDiscordance(genoData, subjName.col,
                     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.
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.

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 the subject

Author(s)

Tushar Bhangale, Cathy Laurie

See Also

GenotypeData, duplicateDiscordanceAcrossDatasets, duplicateDiscordanceProbability

Examples

```
library(GWASdata)
file <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(file)
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
genoData <-  GenotypeData(nc, scanAnnot=scanAnnot)

disc <- duplicateDiscordance(genoData, subjName.col="subjectID")
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,
  subjName.cols, snpName.cols,
  scan.exclude1=NULL, scan.exclude2=NULL, snp.include=NULL,
  verbose=TRUE)
```

Arguments

`genoData1`	`GenotypeData` object containing the first dataset.
`genoData2`	`GenotypeData` object containing the second dataset.
`subjName.cols`	2-element character vector indicating the names of the annotation variables that will be identical for duplicate scans in the two datasets.
`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.
`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.include`	List of SNPs to include in the comparison. Should match the contents of the columns referred to by `snpName.cols`.
`verbose`	Logical value specifying whether to show progress information.

Details

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

Value

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

If no duplicate scans or no common SNPs are found, issues a warning message and returns `NULL`.

Author(s)

Stephanie Gogarten, Jess Shen

See Also

`GenotypeData`, `duplicateDiscordance`, `duplicateDiscordanceProbability`

Examples

```
library(GWASdata)

# dataset 1
file1 <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc1 <- NcdfGenotypeReader(file1)
data(affy_snp_annot)
snpAnnot1 <- SnpAnnotationDataFrame(affy_snp_annot)
data(affy_scan_annot)
scanAnnot1 <- ScanAnnotationDataFrame(affy_scan_annot)
data1 <-  GenotypeData(nc1, snpAnnot=snpAnnot1, scanAnnot=scanAnnot1)

# dataset 2
file2 <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
nc2 <- NcdfGenotypeReader(file2)
data(illumina_snp_annot)
snpAnnot2 <- SnpAnnotationDataFrame(illumina_snp_annot)
data(illumina_scan_annot)
scanAnnot2 <- ScanAnnotationDataFrame(illumina_scan_annot)
data2 <-  GenotypeData(nc2, snpAnnot=snpAnnot2, scanAnnot=scanAnnot2)

discord <- duplicateDiscordanceAcrossDatasets(data1, data2,
             subjName.cols=c("CoriellID", "CoriellID"),
             snpName.cols=c("rsID", "rsID"))
close(data1)
close(data2)
```

---

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.

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=F

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
`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 gender.

`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(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)

blfile <- system.file("extdata", "illumina_bl.nc", package="GWASdata")
blnc <- NcdfIntensityReader(blfile)
blData <- IntensityData(blnc, scanAnnot=scanAnnot)

genofile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
genonc <- NcdfGenotypeReader(genofile)
genoData <- GenotypeData(genonc, scanAnnot=scanAnnot)

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

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

---

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,
                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.
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(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)

data(affy_snp_annot)
snpAnnot <- SnpAnnotationDataFrame(affy_snp_annot)

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

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

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

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

# select samples
scan.sel <- scanAnnot$scanID[scanAnnot$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)
```

---

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, ...)
getScanID(object, ...)
getSex(object, ...)
getSnpVariable(object, varname, ...)
getSnpID(object, ...)
getChromosome(object, ...)
getPosition(object, ...)
getVariable(object, varname, ...)
getGenotype(object, ...)
getQuality(object, ...)
getX(object, ...)
getY(object, ...)
getBAlleleFreq(object, ...)
getLogRRatio(object, ...)
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)
```

```
XchromCode(object)
XYchromCode(object)
YchromCode(object)
MchromCode(object)

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

Arguments

object	Object, possibly derived from or containing NcdfReader-class, ScanAnnotationDataFra class, SnpAnnotationDataFrame-class, ScanAnnotationSQLite-class, or SnpAnnotationSQLite-class.
varname	Name of the variable (single character string, or a character vector for multiple variables).
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.

XchromCode, XYchromCode, YchromCode, and MchromCode return the integer chromosome codes associated with 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, GenotypeData-class, IntensityData-class

---

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)
```

---

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
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, Matt Conomos

See Also

HWExact

Examples

```
# The following example would perform exact Hardy-Weinberg equilibrium testing on all chr

library(GWASdata)
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)

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

# create data object
ncfile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(ncfile)
genoData <-  GenotypeData(nc, scanAnnot=scanAnnot)

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

close(genoData)
```

---

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
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", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(file)
genoData <- GenotypeData(nc)
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", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(file)

# need scan annotation with sex
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
genoData <-  GenotypeData(nc, scanAnnot=scanAnnot)

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

---

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-sib, half-sib, and first-cousins are plotted in orange. ibdAreasDraw overlays relationship areas for IBD analysis on the plot. ibdAssignRelatedness identifies observed relatives.

Usage

```
ibdPlot(k0, k1, alpha=0.05, relation=NULL, color=NULL, rel.lwd=2, ...)

ibdAreasDraw(alpha=.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)
```

Arguments

k0	A vector of k0 values.
k1	A vector of k1 values.
alpha	significance level - finds 100(1-alpha)% prediction intervals for half-sibs and first cousins and 100(1-alpha)% prediction ellipse for full-sibs
relation	A vector of relationships.
color	A vector of colors for (k0,k1) points.
rel.lwd	Line width for theoretical full-sib, half-sib, and first-cousin boundaries.
...	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, half-sib, first cousin, dup & unrelated areas

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.

Theoretical boundary for full-sibs is indicated by ellipse and boundaries for half-sib and first cousin 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 half-sibs and first-cousins, 100(1-alpha)% prediction intervals for k1 are based on assuming normal distribution with known mean and variance.

ibdAreasDraw overlays relationship areas on the plot to help with analyzing observed relationships. For full-sibs, 100(1-alpha)% prediction ellipse is based on assuming bivariate normal distribution with known mean and covariance matrix. For half-sibs and first-cousins, 100(1-alpha)% prediction intervals for k1 are based on assuming normal distribution with known mean and variance.

ibdAssignRelatedness identifies relatives based on their (k0, k1) coordinates.

Value

`ibdAssignRelatedness` returns a vector of relationships with values "Dup"=duplicate, "PO"=parent-offspring, "FS"=full sibling, "HS"=half-sibling-like, "FC"=first cousin, "U"=unrelated, and "Q"=unknown.

Author(s)

Cathy Laurie and Cecelia Laurie

See Also

`relationsMeanVar`

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)
```

---

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", "affy_qxy.nc", package="GWASdata")
nc <- NcdfIntensityReader(file)
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
intenData <- IntensityData(nc, scanAnnot=scanAnnot)
meanInten <- meanIntensityByScanChrom(intenData)
intenMatrix <- meanInten$mean.intensity

# find outliers
outliers <- list()
sex <- scanAnnot$sex
id <- scanAnnot$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)
```

`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,
              chrom.labels = c(1:22,"X","XY","Y","M"),
              ylim = NULL, trunc.lines = TRUE, ...)
```

Arguments

`p`	A vector of p-values.
`chromosome`	A vector containing the integer chromosome ID for each SNP.
`chrom.labels`	A vector of chromosome names to use in the plot.
`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.
`...`	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.

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)`.

Author(s)

Cathy Laurie

See Also

`snpCorrelationPlot`

Examples

```
n <- 1000
pvals <- sample(-log10((1:n)/n), n, replace=TRUE)
chromosome <- c(rep(1,500), rep(2,500))
manhattanPlot(pvals, chromosome, chrom.labels=c(1,2))
```

`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
`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", "affy_qxy.nc", package="GWASdata")
nc <- NcdfIntensityReader(file)
intenData <- IntensityData(nc)

meanInten <- meanIntensityByScanChrom(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, Matt Conomos

See Also

mendelList

Examples

```
library(GWASdata)
data(affy_scan_annot)

# generate trio list
men.list <- with(affy_scan_annot, mendelList(family, subjectID, father, mother, sex, scan

# create genoData object
ncfile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(ncfile)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
genoData <- GenotypeData(nc, 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 ID numbers.
`offspring`	A vector of offspring ID numbers.
`father`	A vector of father ID numbers.
`mother`	A vector of mother ID numbers.
`sex`	A vector to specify whether each scan is male "M" or female "F".
`scanID`	A vector of unique scan identification numbers. These will be used to identify scans in output.
`mendel.list`	An object of class "mendelList".

Details

The lengths of `familyid`, `offspring`, `father`, `mother`, `sex`, and `scanID` must all be identical. 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, Matt Conomos

See Also

`mendelErr`

Examples

```
# get sample annotation
library(GWASdata)
data(affy_scan_annot)

# generate trio list
men.list <- with(affy_scan_annot, mendelList(family, subjectID, father, mother, sex, scan

class(men.list)
# [1] "mendelList"

# convert into a data.frame
men.df <- mendelListAsDataFrame(men.list)

class(men.df)
# [1] "data.frame"
```

---

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
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", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(file)

# need scan annotation with sex
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
genoData <-  GenotypeData(nc, scanAnnot=scanAnnot)

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 gender 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", "affy_geno.nc", package="GWASdata")
nc <- NcdfGenotypeReader(file)

# need scan annotation with sex
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
genoData <-  GenotypeData(nc, scanAnnot=scanAnnot)

missingRate <- missingGenotypeBySnpSex(genoData)
close(genoData)
```

---

ncdfAddData *Write genotypic calls and/or associated metrics to a netCDF file*

Description

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

```
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)
```

`ncdfCreate` *Write genotypic calls and/or associated metrics to a netCDF file.*

Description

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", "LogRRatio")`)
`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

```
library(GWASdata)
data(affy_snp_annot)
ncfile <- tempfile()
ncdfCreate(affy_snp_annot, ncfile, variables="genotype", n.samples=5)
file.remove(ncfile)
```

`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`, `ncdfCreate`, `ncdfAddData`

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)
```

---

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 identity numbers of the family, individual, individual's mother, individual's father and individual's gender (coded as "M" or "F") .

Details

The function pedigreeCheck finds any of a number of possible inconsitencies within pedigree data: single individual families, gender mismatches with mother or father, impossible relationships (either where a child is a parent of self or an individual is both a child and a parent of the same person) and families that consist of two unrelated persons. It also looks for multiple subfamilies within a family.

Value

The output for pedigreeCheck is a list with the following elements:

one.person	A vector of family ids for one-person families
mismatch.sex	A vector of of family ids where sex of mother and/or father is incorrect
impossible.related	A vector of of family ids where either child is mother of self or an individual is both child and mother of same person
duos	A vector of of family ids where 'family' consists of only 2 unrelated persons

`subfamilies.ident`
: A matrix with columns for the family id (of 'families' with multiple subfamilies), subfamily identifier and individual of each person in the identified subfamilies. Note that subfamilies are not identified for any families already identified with problems, and that the individual id's include individuals identified as a mother or father who may not be listed as individuals in the pedigree.

  If no inconsistencies are found, the output is `NULL`.

Note

Subfamilies are not identified for any families already identified with problems, and individual id's in subfamilies may include individual ids listed for a mother or father who may not be listed as individuals in the pedigree.

Author(s)

Cecilia Laurie

See Also

`pedigreeClean`, `pedigreeFindDuplicates`, `pedigreePairwiseRelatedness`

Examples

```
family <- c(1,1,1,2,2,2,3)
individ <- c(1,2,3,4,5,6,7)
mother <- c(0,0,1,0,0,4,0)
father <- c(0,0,2,0,0,5,0)
sex <- c("F","M","F","F","F","M","M")
pedigree <- data.frame(family, individ, mother, father, sex)
pedigreeCheck(pedigree)
#$one.person
#[1] 3

#$mismatch.sex
#[1] 2

#$impossible.related
#NULL

#$subfamilies.ident
#data frame with 0 columns and 0 rows
```

`pedigreeClean` *Basic pedigree data checking*

Description

This function checks a pedigree for completeness and gross errors

Usage

```
pedigreeClean(pedigree, verbose = TRUE)
```

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 identity numbers of the family, individual, individual's mother, individual's father and individual's gender (coded as "M" or "F") .
verbose	Logical value specifying whether or not to show progress information.

Details

The function performs a basic check on pedigree data for gross errors, checking for missing id's, non-integer id's, mis-coded gender, id's of 0 and for individuals that appear as both mothers and fathers.

Value

A list with the following components:

fam.na	A vector of integers containing the row positions of entries with missing family id's
other.na	A vector of integers containing the row positions of entries with missing individual, mother or father id's
cols.not.numeric	A vector of integers containing the row positions of non-numeric id's
rows.sexcode.error	A vector of integers containing the row positions of entries with mis-specified gender
zero.individ	A vector of integers containing the row positions with an id equal to 0.
mofa	A vector containing the id's of individuals appearing as both mothers and fathers

Returns NULL if no errors were found.

Author(s)

Cecelia Laurie

See Also

pedigreeCheck, pedigreeFindDuplicates, pedigreePairwiseRelatedness

Examples

```
family <- c(1,1,1,NA,2,2,2,2)
individ <- c(1,2,3,0,4,5,6,NA)
mother <- c(0,0,1,1,0,0,4,4)
father <- c(0,0,2,2,0,0,5,4)
sex <- c("F","M","F","F","F","F","M",1)
pedigree <- data.frame(family, individ, mother, father, sex)
pedigreeClean(pedigree)

#$fam.na
#[1] 4

#$other.na
```

```
#[1] 8

#$cols.not.numeric
#NULL

#$rows.sexcode.error
#[1] 8

#$zero.individ
#[1] 4

#$mofa
#[1] 2
```

---

pedigreeFindDuplicates

Identify and remove duplicates from a pedigree

Description

pedigreeFindDuplicates identifies duplicates of individuals within a family and checks that pedigree data on duplicates is consistent. pedigreeDeleteDuplicates removes duplicates from a pedigree.

Usage

```
pedigreeFindDuplicates(pedigree, verbose = TRUE)

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 identity numbers of the family, individual, individual's mother, individual's father and individual's gender (coded as "M" or "F") .
duplicates	dataframe with columns "family" (family id) and "individ" (individual id)
verbose	Logical value specifying whether or not to show progress information.

Details

The output of pedigreeFindDuplicates can be provided to pedigreeDeleteDuplicates in order to generate a new pedigree with duplicates removed.

Value

The output of pedigreeFindDuplicates is list containing two dataframes:

dups.mismatch
: A dataframe containing the family id, individual id and number of copies for any duplicates with mismatching pedigree data

`dups.match` A dataframe containing the family id, individual id and number of copies for any duplicates with matching pedigree data

The output of `pedigreeDeleteDuplicates` is a pedigree identical to `pedigree`, but with duplicates removed.

Author(s)

Cecilia Laurie

See Also

`pedigreeClean`, `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)
duplicates <- pedigreeFindDuplicates(pedigree)
pedigree.no.dups <- pedigreeDeleteDuplicates(pedigree, duplicates$dups.match)
```

---

pedigreePairwiseRelatedness

Calculate theoretical pairwise relatedness values from pedigrees

---

Description

This function calculates the pairwise relatedness values from pedigree data.

Usage

```
pedigreePairwiseRelatedness(pedigree, use.any.ids = FALSE)
```

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 identity numbers of the family, individual, individual's mother, individual's father and individual's gender (coded as "M" or "F") .

`use.any.ids` A logical value specifying whether pairs of individuals should be created using only id's listed in the "individ" column (if `FALSE` or if pairs should be created using any id's contained in "individ", "mother" or "father" columns.

Details

The function assumes (and checks) that there are no one person families, no mismatched mother/father sexes and no impossible relationships. Relatedness is not calculated for inbred families.

Value

A list with the following components:

`inbred.fam`	A vector of id's of families with inbreeding (to be handled by hand)
`relativeprs`	A dataframe with columns "Individ1", "Individ2", "relation", "kinship coefficient" and "family" containing the id's of the pair of individuals, the relationship between the individuals if closely related (possible values are "UN" = unrelated, "PO" = parent/offspring, "FS" = full siblings, "HS" = half siblings, and "FC" = first cousins), kinship coefficient and family id.

Author(s)

Cecilia Laurie

See Also

`pedigreeClean`, `pedigreeCheck`, `pedigreeFindDuplicates`

Examples

```
family <- c(1,1,1,1,2,2,2,2)
individ <- c(1,2,3,4,5,6,7,8)
mother <- c(0,0,1,1,0,0,5,5)
father <- c(0,0,2,2,0,0,6,0)
sex <- c("F","M","F","F","F","M","M","M")
pedigree <- data.frame(family, individ, mother, father, sex)
pedigreePairwiseRelatedness(pedigree)

#$inbred.fam
#NULL

#$relativeprs
#   Individ1 Individ2 relation kinship family
#1         1        2        U   0.000      1
#2         1        3       PO   0.250      1
#3         1        4       PO   0.250      1
#4         2        3       PO   0.250      1
#5         2        4       PO   0.250      1
#6         3        4       FS   0.250      1
#11        5        6        U   0.000      2
#21        5        7       PO   0.250      2
#31        5        8       PO   0.250      2
#51        6        7       PO   0.250      2
#61        6        8        U   0.000      2
#8         7        8       HS   0.125      2
```

`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, code=NULL,
  plotY=FALSE, hg.build=c("hg18", "hg19"), ...)
```

Arguments

scan.ids	A vector containing the sample indices of the plots.
intenData	IntensityData object, must contain 'BAlleleFreq' and 'LogRRatio'
code	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
...	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

See Also

pseudoautosomal, IntensityData, GenotypeData, BAFfromGenotypes

Examples

```
library(GWASdata)
data(illumina_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
blfile <- system.file("extdata", "illumina_bl.nc", package="GWASdata")
blnc <- NcdfIntensityReader(blfile)
intenData <-  IntensityData(blnc, scanAnnot=scanAnnot)

scanID <- getScanID(scanAnnot, 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` chromsome (X or Y)

`region` region (XTR, PAR1, or PAR2)

`start.base` starting base position of region

`end.base` ending base position of region

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

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, sub = NULL, ...)
```

Arguments

pval	Vector of p-values
truncate	A logical value indicating whether the y-axis should be truncted to the same range as the x-axis.
sub	A character string to print under the x-axis.
...	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 sub=NULL (the default), the genomic inflation factor lambda is calculated from the p-values and printed.

Author(s)

Cathy Laurie, Matt Conomos

Examples

```
pvals <- seq(0, 1, 0.001)
qqPlot(pvals)
```

---

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", "affy_qxy.nc", package="GWASdata")
qualnc <- NcdfIntensityReader(qualfile)
# need scan annotation with sex
data(affy_scan_annot)
scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
qualData <- IntensityData(qualnc, scanAnnot=scanAnnot)

genofile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
genonc <- NcdfGenotypeReader(genofile)
genoData <- GenotypeData(genonc, scanAnnot=scanAnnot)

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", "affy_qxy.nc", package="GWASdata")
qualnc <- NcdfIntensityReader(qualfile)
qualData <- IntensityData(qualnc)

genofile <- system.file("extdata", "affy_geno.nc", package="GWASdata")
genonc <- NcdfGenotypeReader(genofile)
genoData <- GenotypeData(genonc)

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)
```

`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
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 0 indicates an AA genotype, 1 is AB, 2 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
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)
```

---

snpCorrelationPlot *SNP correlation plot*

Description

Plots SNP correlation versus chromosome.

Usage

```
snpCorrelationPlot(correlations, chromosome,
                   chrom.labels = c(1:22,"X","XY","Y","M"),
                   ylim=c(0,1), ylab = "abs(correlation)", ...)
```

Arguments

correlations	A vector of correlations.
chromosome	A vector containing the integer chromosome ID for each SNP.
chrom.labels	A vector of chromosome names to use in the plot.
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. SNPs are evenly spaced along the X axis, so correlations should be in order of position on the chromosome.

Author(s)

Cathy Laurie

See Also

manhattanPlot

Examples

```
correlations <- sample(0.01*(0:100), 100, replace=TRUE)
chromosome <- c(rep(1,50), rep(2,50))
snpCorrelationPlot(correlations, chromosome, chrom.labels=c(1,2))
```

Index