

Package ‘BGData’

December 8, 2021

Version 2.4.0

License MIT + file LICENSE

Title A Suite of Packages for Analysis of Big Genomic Data

Description An umbrella package providing a phenotype/genotype data structure and scalable and efficient computational methods for large genomic datasets in combination with several other packages: 'BEDMatrix', 'LinkedMatrix', and 'symDMatrix'.

URL <https://github.com/QuantGen/BGData>

BugReports <https://github.com/QuantGen/BGData/issues>

Depends R (>= 3.0.2), BEDMatrix (>= 1.4.0), LinkedMatrix (>= 1.3.0), symDMatrix (>= 2.0.0)

Imports methods, parallel, crochet (>= 2.1.0), bigmemory, synchronicity, ff, bit

Suggests data.table (>= 1.9.6), lme4, SKAT, testthat

NeedsCompilation yes

Author Gustavo de los Campos [aut],
Alexander Grueneberg [aut, cre],
Paulino Perez [ctb],
Ana Vazquez [ctb]

Maintainer Alexander Grueneberg <cran@agrueneberg.info>

Repository CRAN

Date/Publication 2021-12-08 15:30:02 UTC

R topics documented:

BGData-package	2
as.BGData	3
BGData	4
BGData-class	5
chunkedApply	6
chunkedMap	7

file-backed-matrices	8
findRelated	9
FWD	10
geno	11
geno-class	12
getG	12
getG_symDMatrix	14
GWAS	16
load.BGData	18
multi-level-parallelism	18
orderedMerge	19
preprocess	19
readRAW	20
segments	23
summarize	24
Index	26

 BGData-package

A Suite of Packages for Analysis of Big Genomic Data

Description

Modern genomic datasets are big (large n), high-dimensional (large p), and multi-layered. The challenges that need to be addressed are memory requirements and computational demands. Our goal is to develop software that will enable researchers to carry out analyses with big genomic data within the R environment.

Details

We have identified several approaches to tackle those challenges within R:

- File-backed matrices: The data is stored in on the hard drive and users can read in smaller chunks when they are needed.
- Linked arrays: For very large datasets a single file-backed array may not be enough or convenient. A linked array is an array whose content is distributed over multiple file-backed nodes.
- Multiple dispatch: Methods are presented to users so that they can treat these arrays pretty much as if they were RAM arrays.
- Multi-level parallelism: Exploit multi-core and multi-node computing.
- Inputs: Users can create these arrays from standard formats (e.g., PLINK .bed).

The BGData package is an umbrella package that comprises several packages: `BEDMatrix`, `LinkedMatrix`, and `symDMatrix`. It features scalable and efficient computational methods for large genomic datasets such as genome-wide association studies (GWAS) or genomic relationship matrices (G matrix). It also contains a container class called `BGData` that holds genotypes, sample information, and variant information.

Example dataset

The `extdata` folder contains example files that were generated from the 250k SNP and phenotype data in [Atwell et al. \(2010\)](#). Only the first 300 SNPs of chromosome 1, 2, and 3 were included to keep the size of the example dataset small. `PLINK` was used to convert the data to `.bed` and `.raw` files. FT10 has been chosen as a phenotype and is provided as an [alternate phenotype file](#). The file is intentionally shuffled to demonstrate that the additional phenotypes are put in the same order as the rest of the phenotypes.

See Also

[BEDMatrix-package](#), [LinkedMatrix-package](#), and [symdMatrix-package](#) for an introduction to the respective packages.

[file-backed-matrices](#) for more information on file-backed matrices. [multi-level-parallelism](#) for more information on multi-level parallelism.

 as.BGData

Convert Other Objects to BGData Objects

Description

Converts other objects to `BGData` objects by loading supplementary phenotypes and map files referenced by the object to be used for the sample information and variant information, respectively.

Currently supported are `BEDMatrix` objects, plain or nested in `ColumnLinkedMatrix` objects.

Usage

```
as.BGData(x, alternatePhenotypeFile = NULL, ...)

## S3 method for class 'BEDMatrix'
as.BGData(x, alternatePhenotypeFile = NULL, ...)

## S3 method for class 'ColumnLinkedMatrix'
as.BGData(x, alternatePhenotypeFile = NULL,
          ...)

## S3 method for class 'RowLinkedMatrix'
as.BGData(x, alternatePhenotypeFile = NULL,
          ...)
```

Arguments

<code>x</code>	An object. Currently supported are <code>BEDMatrix</code> objects, plain or nested in <code>ColumnLinkedMatrix</code> objects.
<code>alternatePhenotypeFile</code>	Path to an alternate phenotype file .
<code>...</code>	Additional arguments to the <code>read.table</code> or <code>fread</code> call (if <code>data.table</code> package is installed) call to parse the alternate pheno file.

Details

The .ped and .raw formats only allows for a single phenotype. If more phenotypes are required it is possible to store them in an **alternate phenotype file**. The path to such a file can be provided with `alternatePhenotypeFile` and will be merged with the existing sample information. The first and second columns of that file must contain family and within-family IDs, respectively.

For `BEDMatrix` objects: If a .fam file (which corresponds to the first six columns of a .ped or .raw file) of the same name and in the same directory as the .bed file exists, the sample information will be populated with the data stored in that file. Otherwise a stub that only contains an IID column populated with the rownames of `geno(x)` will be generated. The same will happen for a .bim file for the variant information.

For `ColumnLinkedMatrix` objects: See the case for `BEDMatrix` objects, but only the .fam file of the first node of the `LinkedMatrix` will be read and used for the sample information, and the .bim files of all nodes will be combined and used for the variant information.

Value

A `BGData` object.

See Also

`readRAW()` to convert text files to `BGData` objects. [BGData-class](#), [BEDMatrix-class](#), [ColumnLinkedMatrix-class](#) for more information on the above mentioned classes. [read.table](#) and [fread](#) to learn more about extra arguments that can be passed via `...`

Examples

```
# Path to example data
path <- system.file("extdata", package = "BGData")

# Convert a single BEDMatrix object to a BGData object
chr1 <- BEDMatrix::BEDMatrix(paste0(path, "/chr1.bed"))
bg1 <- as.BGData(chr1)

# Convert multiple BEDMatrix objects in a ColumnLinkedMatrix to a BGData object
chr2 <- BEDMatrix::BEDMatrix(paste0(path, "/chr2.bed"))
chr3 <- BEDMatrix::BEDMatrix(paste0(path, "/chr3.bed"))
clm <- ColumnLinkedMatrix(chr1, chr2, chr3)
bg2 <- as.BGData(clm)

# Load additional (alternate) phenotypes
bg3 <- as.BGData(clm, alternatePhenotypeFile = paste0(path, "/pheno.txt"))
```

BGData

Creates a New BGData Instance

Description

This function constructs a new `BGData` object.

Usage

```
BGData(geno, pheno = NULL, map = NULL)
```

Arguments

geno	A geno object that contains genotypes.
pheno	A data.frame that contains sample information (including phenotypes). A stub that only contains a sample_id column populated with either the rownames of geno or a sequence starting with sample_ will be generated if NULL
map	A data.frame that contains variant information. A stub that only contains a variant_id column populated with either the colnames of geno or a sequence starting with variant_ will be generated if NULL

See Also

[BGData-class](#) and [geno-class](#) for more information on the above mentioned classes.

BGData-class

Container for Phenotype and Genotype Data

Description

The BGData class is a container for genotypes, sample information, and variant information. The class is inspired by the .bed/.fam/.bim (binary) and .ped/.fam/.map (text) phenotype/genotype file formats of **PLINK**. It is used by several functions of this package such as GWAS for performing a Genome Wide Association Study or getG for calculating a genomic relationship matrix.

Details

There are several ways to create an instance of this class:

- from arbitrary phenotype/genotype data using the BGData constructor function.
- from a .bed file using as.BGData and BEDMatrix.
- from a previously saved BGData object using load.BGData.
- from multiple files (even a mixture of different file types) using LinkedMatrix.
- from a .raw file (or a .ped-like file) using readRAW, readRAW_matrix, or readRAW_big.matrix.

A .ped file can be recoded to a .raw file in **PLINK** using `plink --file myfile --recodeA`, or converted to a .bed file using `plink --file myfile --make-bed`. Conversely, a .bed file can be transformed back to a .ped file using `plink --bfile myfile --recode` or to a .raw file using `plink --bfile myfile --recodeA` without losing information.

Accessors

In the following code snippets, `x` is a `BGData` object.

`geno(x)`, `geno(x) <- value`: Get or set genotypes.

`pheno(x)`, `pheno(x) <- value`: Get or set sample information.

`map(x)`, `map(x) <- value`: Get or set variant information.

See Also

[BGData](#), [as.BGData](#), [load.BGData](#), [readRAW](#) to create `BGData` objects.

[LinkedMatrix-class](#) and [BEDMatrix-class](#) for more information on the above mentioned classes.

Examples

```
X <- matrix(data = rnorm(100), nrow = 10, ncol = 10)
Y <- data.frame(y = runif(10))
MAP <- data.frame(means = colMeans(X), freqNA = colMeans(is.na(X)))
DATA <- BGData(geno = X, pheno = Y, map = MAP)

dim(geno(DATA))
head(pheno(DATA))
head(map(DATA))
```

chunkedApply

Applies a Function on Each Row or Column of a File-Backed Matrix

Description

Similar to `apply`, but designed for file-backed matrices. The function brings chunks of an object into physical memory by taking subsets, and applies a function on either the rows or the columns of the chunks using an optimized version of `apply`. If `nCores` is greater than 1, the function will be applied in parallel using `mclapply`. In that case the subsets of the object are taken on the slaves.

Usage

```
chunkedApply(X, MARGIN, FUN, i = seq_len(nrow(X)),
             j = seq_len(ncol(X)), chunkSize = 5000L,
             nCores = getOption("mc.cores", 2L), verbose = FALSE, ...)
```

Arguments

<code>X</code>	A file-backed matrix, typically the genotypes of a <code>BGData</code> object.
<code>MARGIN</code>	The subscripts which the function will be applied over. 1 indicates rows, 2 indicates columns.
<code>FUN</code>	The function to be applied.

<code>i</code>	Indicates which rows of <code>X</code> should be used. Can be integer, boolean, or character. By default, all rows are used.
<code>j</code>	Indicates which columns of <code>X</code> should be used. Can be integer, boolean, or character. By default, all columns are used.
<code>chunkSize</code>	The number of rows or columns of <code>X</code> that are brought into physical memory for processing per core. If <code>NULL</code> , all elements in <code>i</code> or <code>j</code> are used. Defaults to 5000.
<code>nCores</code>	The number of cores (passed to <code>mclapply</code>). Defaults to the number of cores as detected by <code>detectCores</code> .
<code>verbose</code>	Whether progress updates will be posted. Defaults to <code>FALSE</code> .
<code>...</code>	Additional arguments to be passed to the apply like function.

See Also

[file-backed-matrices](#) for more information on file-backed matrices. [multi-level-parallelism](#) for more information on multi-level parallelism. [BGData-class](#) for more information on the `BGData` class.

Examples

```
# Restrict number of cores to 1 on Windows
if (.Platform$OS.type == "windows") {
  options(mc.cores = 1)
}

# Load example data
bg <- BGData:::loadExample()

# Compute standard deviation of columns
chunkedApply(X = geno(bg), MARGIN = 2, FUN = sd)
```

 chunkedMap

Applies a Function on Each Chunk of a File-Backed Matrix

Description

Similar to `lapply`, but designed for file-backed matrices. The function brings chunks of an object into physical memory by taking subsets, and applies a function on them. If `nCores` is greater than 1, the function will be applied in parallel using `mclapply`. In that case the subsets of the object are taken on the slaves.

Usage

```
chunkedMap(X, FUN, i = seq_len(nrow(X)), j = seq_len(ncol(X)),
  chunkBy = 2L, chunkSize = 5000L, nCores = getOption("mc.cores",
  2L), verbose = FALSE, ...)
```

Arguments

<code>X</code>	A file-backed matrix, typically the genotypes of a <code>BGData</code> object.
<code>FUN</code>	The function to be applied on each chunk.
<code>i</code>	Indicates which rows of <code>X</code> should be used. Can be integer, boolean, or character. By default, all rows are used.
<code>j</code>	Indicates which columns of <code>X</code> should be used. Can be integer, boolean, or character. By default, all columns are used.
<code>chunkBy</code>	Whether to extract chunks by rows (1) or by columns (2). Defaults to columns (2).
<code>chunkSize</code>	The number of rows or columns of <code>X</code> that are brought into physical memory for processing per core. If <code>NULL</code> , all elements in <code>i</code> or <code>j</code> are used. Defaults to 5000.
<code>nCores</code>	The number of cores (passed to <code>mclapply</code>). Defaults to the number of cores as detected by <code>detectCores</code> .
<code>verbose</code>	Whether progress updates will be posted. Defaults to <code>FALSE</code> .
<code>...</code>	Additional arguments to be passed to the <code>apply</code> like function.

See Also

[file-backed-matrices](#) for more information on file-backed matrices. [multi-level-parallelism](#) for more information on multi-level parallelism. [BGData-class](#) for more information on the `BGData` class.

Examples

```
# Restrict number of cores to 1 on Windows
if (.Platform$OS.type == "windows") {
  options(mc.cores = 1)
}

# Load example data
bg <- BGData:::loadExample()

# Compute column sums of each chunk
chunkedMap(X = geno(bg), FUN = colSums)
```

file-backed-matrices *File-Backed Matrices*

Description

Functions with the `chunkSize` parameter work best with file-backed matrices such as `BEDMatrix` objects. To avoid loading the whole, potentially very large matrix into memory, these functions will load chunks of the file-backed matrix into memory and perform the operations on one chunk at a time. The size of the chunks is determined by the `chunkSize` parameter. Care must be taken to not set `chunkSize` too high to avoid memory shortage, particularly when combined with parallel computing.

See Also

[BEDMatrix-class](#) as an example of a file-backed matrix.

findRelated	<i>Find related individuals in a relationship matrix</i>
-------------	--

Description

Find related individuals in a relationship matrix.

Usage

```
findRelated(x, ...)

## S3 method for class 'matrix'
findRelated(x, cutoff = 0.03, ...)

## S3 method for class 'symDMatrix'
findRelated(x, cutoff = 0.03, verbose = FALSE,
  ...)
```

Arguments

x	A matrix-like object with dimnames.
...	Additional arguments for methods.
cutoff	The cutoff between 0 and 1 for related individuals to be included in the output. Defaults to 0.03.
verbose	Whether progress updates will be posted. Defaults to FALSE.

Value

A vector of names or indices of related individuals.

Methods (by class)

- `matrix`: Find related individuals in matrices
- `symDMatrix`: Find related individuals in `symDMatrix` objects

Examples

```
# Load example data
bg <- BGData:::loadExample()

G <- getG(geno(bg))
findRelated(G)
```

FWD

*Performs Forward Regressions***Description**

Performs forward regression of y on the columns of X . Predictors are added, one at a time, each time adding the one that produces the largest reduction in the residual sum of squares (RSS). The function returns estimates and summaries for the entire forward search. This function performs a similar search than that of `step(<, direction='forward')`, however, `FWD()` is optimized for computational speed for linear models with very large sample size. To achieve fast computations, the software first computes the sufficient statistics $X'X$ and $X'y$. At each step, the function first finds the predictor that produces the largest reduction in the sum of squares (this can be derived from $X'X$, $X'y$ and the current solution of effects), and then updates the estimates of effects for the resulting model using Gauss Seidel iterations performed on the linear system $(X'X)b=X'y$, iterating only over the elements of b that are active in the model.

Usage

```
FWD(y, X, df = 20, tol = 1e-7, maxIter = 1000, centerImpute = TRUE,
    verbose = TRUE)
```

Arguments

<code>y</code>	The response vector (numeric $n \times 1$).
<code>X</code>	An ($n \times p$) numeric matrix. Columns are the features (aka predictors) considered in the forward search. The rows of X must be matched to the entries of y .
<code>df</code>	Defines the maximum number of predictors to be included in the model. For complete forward search, set <code>df = ncol(X)</code> .
<code>tol</code>	A tolerance parameter to control when to stop the Gauss Seidel algorithm.
<code>maxIter</code>	The maximum number of iterations for the Gauss Seidel algorithm (only used when the algorithm is not stopped by the tolerance parameter).
<code>centerImpute</code>	Whether to center the columns of X and impute the missing values with the column means.
<code>verbose</code>	Use <code>verbose = TRUE</code> to print summaries of the forward search.

Value

A list with two entries:

- `B`: ($p \times df + 1$) includes the estimated effects for each predictor (rows) at each step of the forward search (`df`, in columns).
- `path`: A data frame providing the order in which variables were added to the model (`variable`) and statistics for each step of the forward search (RSS, `LogLik`, VARE (the residual variance), DF, AIC, and BIC).

geno	<i>Getting/Setting Genotypes, Sample Information, and Variant Information</i>
------	---

Description

A set of generic functions for getting/setting the genotypes, sample information, and variant information.

Usage

```
geno(x)
geno(x) <- value

pheno(x)
pheno(x) <- value

map(x)
map(x) <- value
```

Arguments

x	The object from/on which to get/set genotypes, sample information, and variant information. Typically a BGData object.
value	Typically a geno object for the geno setter. Typically a data.frame object for the pheno setter. Typically a data.frame object for the map setter.

See Also

- [BGData-class](#)
- [geno-class](#)

Examples

```
# Load example data
bg <- BGData:::loadExample()

# Access genotypes
geno(bg)

# Access sample information
pheno(bg)

# Access variant information
map(bg)
```

 geno-class

An Abstract S4 Class Union of Matrix-Like Types

Description

geno is a class union of several matrix-like types, many of them suitable for very large datasets. Currently supported are `LinkedMatrix`, `BEDMatrix`, `big.matrix`, `ff_matrix`, and `matrix`.

See Also

[LinkedMatrix-class](#), [BEDMatrix-class](#), [big.matrix-class](#), [ff](#), and [matrix](#) for more information on each matrix-like type.

[BGData-class](#) for more information on the BGData class, in particular its geno accessor that accepts geno objects.

 getG

Computes a Genomic Relationship Matrix

Description

Computes a positive semi-definite symmetric genomic relation matrix $G=XX'$ offering options for centering and scaling the columns of X beforehand.

Usage

```
getG(X, center = TRUE, scale = TRUE, impute = TRUE, scaleG = TRUE,
      minVar = 1e-05, i = seq_len(nrow(X)), j = seq_len(ncol(X)), i2 = NULL,
      chunkSize = 5000L, nCores = getOption("mc.cores", 2L), verbose = FALSE)
```

Arguments

X	A matrix-like object, typically the genotypes of a BGData object.
center	Either a logical value or a numeric vector of length equal to the number of columns of X. Numeric vector required if i2 is used. If FALSE, no centering is done. Defaults to TRUE.
scale	Either a logical value or a numeric vector of length equal to the number of columns of X. Numeric vector required if i2 is used. If FALSE, no scaling is done. Defaults to TRUE.
impute	Indicates whether missing values should be imputed. Defaults to TRUE.
scaleG	Whether XX' should be scaled. Defaults to TRUE.
minVar	Columns with variance lower than this value will not be used in the computation (only if scale is not FALSE).

i	Indicates which rows of X should be used. Can be integer, boolean, or character. By default, all rows are used.
j	Indicates which columns of X should be used. Can be integer, boolean, or character. By default, all columns are used.
i2	Indicates which rows should be used to compute a block of the genomic relationship matrix. Will compute XY' where X is determined by i and j and Y by i2 and j. Can be integer, boolean, or character. If NULL, the whole genomic relationship matrix XX' is computed. Defaults to NULL.
chunkSize	The number of columns of X that are brought into physical memory for processing per core. If NULL, all columns of X are used. Defaults to 5000.
nCores	The number of cores (passed to <code>mclapply</code>). Defaults to the number of cores as detected by <code>detectCores</code> .
verbose	Whether progress updates will be posted. Defaults to FALSE.

Details

If `center = FALSE`, `scale = FALSE` and `scaleG = FALSE`, `getG` produces the same outcome than `tcrossprod`.

Value

A positive semi-definite symmetric numeric matrix.

See Also

[file-backed-matrices](#) for more information on file-backed matrices. [multi-level-parallelism](#) for more information on multi-level parallelism. [BGData-class](#) for more information on the BGData class.

Examples

```
# Restrict number of cores to 1 on Windows
if (.Platform$OS.type == "windows") {
  options(mc.cores = 1)
}

# Load example data
bg <- BGData:::loadExample()

# Compute a scaled genomic relationship matrix from centered and scaled
# genotypes
g1 <- getG(X = geno(bg))

# Disable scaling of G
g2 <- getG(X = geno(bg), scaleG = FALSE)

# Disable centering of genotypes
g3 <- getG(X = geno(bg), center = FALSE)
```

```

# Disable scaling of genotypes
g4 <- getG(X = geno(bg), scale = FALSE)

# Provide own scales
scales <- chunkedApply(X = geno(bg), MARGIN = 2, FUN = sd)
g4 <- getG(X = geno(bg), scale = scales)

# Provide own centers
centers <- chunkedApply(X = geno(bg), MARGIN = 2, FUN = mean)
g5 <- getG(X = geno(bg), center = centers)

# Only use the first 50 individuals (useful to account for population structure)
g6 <- getG(X = geno(bg), i = 1:50)

# Only use the first 100 markers (useful to ignore some markers)
g7 <- getG(X = geno(bg), j = 1:100)

# Compute unscaled G matrix by combining blocks of  $XX_{i2}'$  where  $XX_{i2}$  is
# a horizontal partition of X. This is useful for distributed computing as each
# block can be computed in parallel. Centers and scales need to be precomputed.
block1 <- getG(X = geno(bg), i2 = 1:100, center = centers, scale = scales)
block2 <- getG(X = geno(bg), i2 = 101:199, center = centers, scale = scales)
g8 <- cbind(block1, block2)

# Compute unscaled G matrix by combining blocks of  $X_{i1}X_{i2}'$  where both
#  $X_{i1}$  and  $X_{i2}$  are horizontal partitions of X. Similarly to the example
# above, this is useful for distributed computing, in particular to compute
# very large G matrices. Centers and scales need to be precomputed. This
# approach is similar to the one taken by the symDMatrix package, but the
# symDMatrix package adds memory-mapped blocks, only stores the upper side of
# the triangular matrix, and provides a type that allows for indexing as if the
# full G matrix is in memory.
block11 <- getG(X = geno(bg), i = 1:100, i2 = 1:100, center = centers, scale = scales)
block12 <- getG(X = geno(bg), i = 1:100, i2 = 101:199, center = centers, scale = scales)
block21 <- getG(X = geno(bg), i = 101:199, i2 = 1:100, center = centers, scale = scales)
block22 <- getG(X = geno(bg), i = 101:199, i2 = 101:199, center = centers, scale = scales)
g9 <- rbind(
  cbind(block11, block12),
  cbind(block21, block22)
)

```

getG_symDMatrix

Computes a Very Large Genomic Relationship Matrix

Description

Computes a positive semi-definite symmetric genomic relation matrix $G=XX'$ offering options for centering and scaling the columns of X beforehand.

Usage

```
getG_symDMatrix(X, center = TRUE, scale = TRUE, impute = TRUE, scaleG = TRUE,
  minVar = 1e-05, blockSize = 5000L,
  folderOut = paste0("symDMatrix_", randomString()), vmode = "double",
  i = seq_len(nrow(X)), j = seq_len(ncol(X)), chunkSize = 5000L,
  nCores = getOption("mc.cores", 2L), verbose = FALSE)
```

Arguments

<code>X</code>	A matrix-like object, typically the genotypes of a BGData object.
<code>center</code>	Either a logical value or a numeric vector of length equal to the number of columns of <code>X</code> . If <code>FALSE</code> , no centering is done. Defaults to <code>TRUE</code> .
<code>scale</code>	Either a logical value or a numeric vector of length equal to the number of columns of <code>X</code> . If <code>FALSE</code> , no scaling is done. Defaults to <code>TRUE</code> .
<code>impute</code>	Indicates whether missing values should be imputed. Defaults to <code>TRUE</code> .
<code>scaleG</code>	<code>TRUE/FALSE</code> whether <code>xx'</code> must be scaled.
<code>minVar</code>	Columns with variance lower than this value will not be used in the computation (only if <code>scale</code> is not <code>FALSE</code>).
<code>blockSize</code>	The number of rows and columns of each block. If <code>NULL</code> , a single block of the same length as <code>i</code> will be created. Defaults to <code>5000</code> .
<code>folderOut</code>	The path to the folder where to save the <code>symDMatrix</code> object. Defaults to a random string prefixed with <code>"symDMatrix_"</code> .
<code>vmode</code>	<code>vmode</code> of <code>ff</code> objects.
<code>i</code>	Indicates which rows of <code>X</code> should be used. Can be integer, boolean, or character. By default, all rows are used.
<code>j</code>	Indicates which columns of <code>X</code> should be used. Can be integer, boolean, or character. By default, all columns are used.
<code>chunkSize</code>	The number of columns of <code>X</code> that are brought into physical memory for processing per core. If <code>NULL</code> , all columns of <code>X</code> are used. Defaults to <code>5000</code> .
<code>nCores</code>	The number of cores (passed to <code>mclapply</code>). Defaults to the number of cores as detected by <code>detectCores</code> .
<code>verbose</code>	Whether progress updates will be posted. Defaults to <code>FALSE</code> .

Details

Even very large genomic relationship matrices are supported by partitioning `X` into blocks and calling `getG` on these blocks. This function performs the block computations sequentially, which may be slow. In an HPC environment, performance can be improved by manually distributing these operations to different nodes.

Value

A `symDMatrix` object.

See Also

[multi-level-parallelism](#) for more information on multi-level parallelism. [symDMatrix-class](#) and [BGData-class](#) for more information on the BGData class. [getG](#) to learn more about the underlying method.

 GWAS

Performs Single Marker Regressions Using BGData Objects

Description

Implements single marker regressions. The regression model includes all the covariates specified in the right-hand-side of the formula plus one column of the genotypes at a time. The data from the association tests is obtained from a BGData object.

Usage

```
GWAS(formula, data, method = "lsfit", i = seq_len(nrow(geno(data))),
      j = seq_len(ncol(geno(data))), chunkSize = 5000L,
      nCores = getOption("mc.cores", 2L), verbose = FALSE, ...)
```

Arguments

formula	The formula for the GWAS model without the variant, e.g. $y \sim 1$ or $y \sim \text{factor}(\text{sex}) + \text{age}$. The variables included in the formula must be column names in the sample information of the BGData object.
data	A BGData object.
method	The regression method to be used. Currently, the following methods are implemented: rayOLS (see below), lsfit, lm, lm.fit, glm, lmer, and SKAT. Defaults to lsfit.
i	Indicates which rows of the genotypes should be used. Can be integer, boolean, or character. By default, all rows are used.
j	Indicates which columns of the genotypes should be used. Can be integer, boolean, or character. By default, all columns are used.
chunkSize	The number of columns of the genotypes that are brought into physical memory for processing per core. If NULL, all elements in j are used. Defaults to 5000.
nCores	The number of cores (passed to mclapply). Defaults to the number of cores as detected by detectCores.
verbose	Whether progress updates will be posted. Defaults to FALSE.
...	Additional arguments for chunkedApply and regression method.

Details

The rayOLS method is a regression through the origin that can only be used with a $y \sim 1$ formula, i.e. it only allows for one quantitative response variable y and one variant at a time as an explanatory variable (the variant is not included in the formula, hence 1 is used as a dummy). If covariates are needed, consider preadjustment of y . Among the provided methods, it is by far the fastest.

Some regression methods may require the data to not contain columns with variance 0 or too many missing values. We suggest running `summarize` to detect variants that do not clear the desired minor-allele frequency and rate of missing genotype calls, and filtering these variants out using the `j` parameter of the GWAS function (see example below).

Value

The same matrix that would be returned by `coef(summary(model))`.

See Also

[file-backed-matrices](#) for more information on file-backed matrices. [multi-level-parallelism](#) for more information on multi-level parallelism. [BGData-class](#) for more information on the BGData class. [lsfit](#), [lm](#), [lm.fit](#), [glm](#), [lmer](#), and [SKAT](#) for more information on regression methods.

Examples

```
# Restrict number of cores to 1 on Windows
if (.Platform$OS.type == "windows") {
  options(mc.cores = 1)
}

# Load example data
bg <- BGData::loadExample()

# Detect variants that do not pass MAF and missingness thresholds
summaries <- summarize(geno(bg))
maf <- ifelse(summaries$allele_freq > 0.5, 1 - summaries$allele_freq,
  summaries$allele_freq)
exclusions <- maf < 0.01 | summaries$freq_na > 0.05

# Perform a single marker regression
res1 <- GWAS(formula = FT10 ~ 1, data = bg, j = !exclusions)

# Draw a Manhattan plot
plot(-log10(res1[, 4]))

# Use lm instead of lsfit (the default)
res2 <- GWAS(formula = FT10 ~ 1, data = bg, method = "lm", j = !exclusions)

# Use glm instead of lsfit (the default)
y <- pheno(bg)$FT10
pheno(bg)$FT10.01 <- y > quantile(y, 0.8, na.rm = TRUE)
res3 <- GWAS(formula = FT10.01 ~ 1, data = bg, method = "glm", j = !exclusions)

# Perform a single marker regression on the first 50 markers (useful for
```

```
# distributed computing)
res4 <- GWAS(formula = FT10 ~ 1, data = bg, j = 1:50)
```

load.BGData	<i>Loads BGData (and Other) Objects from .RData Files</i>
-------------	---

Description

This function is similar to `load`, but also initializes the different types of objects that can be used as genotypes in a `BGData` object.

Currently supported are `ff_matrix`, `big.matrix`, and `BEDMatrix` objects. If the object is of type `LinkedMatrix`, all nodes will be initialized with their appropriate method.

Usage

```
load.BGData(file, envir = parent.frame())
```

Arguments

<code>file</code>	The name of the <code>.RData</code> file to be loaded.
<code>envir</code>	The environment where to load the data.

See Also

[BGData-class](#), [ff](#), [big.matrix-class](#), [BEDMatrix-class](#), and [LinkedMatrix-class](#) for more information on the above mentioned classes.

multi-level-parallelism	<i>Multi-Level Parallelism</i>
-------------------------	--------------------------------

Description

Functions with the `nCores`, `i`, and `j` parameters provide capabilities for both parallel and distributed computing.

For parallel computing, `nCores` determines the number of cores the code is run on. Memory usage can be an issue for higher values of `nCores` as R is not particularly memory-efficient. As a rule of thumb, at least around $(nCores * object_size(chunk)) + object_size(result)$ MB of total memory will be needed for operations on file-backed matrices, not including potential copies of your data that might be created (for example `lsfit` runs `cbind(1,X)`). `i` and `j` can be used to include or exclude certain rows or columns. Internally, the `mclapply` function is used and therefore parallel computing will not work on Windows machines.

For distributed computing, `i` and `j` determine the subset of the input matrix that the code runs on. In an HPC environment, this can be used not just to include or exclude certain rows or columns, but also to partition the task among many nodes rather than cores. Scheduler-specific code and code to aggregate the results need to be written by the user. It is recommended to set `nCores` to 1 as nodes are often cheaper than cores.

See Also

[mclapply](#) to learn more about the function used to implement parallel computing. [detectCores](#) to detect the number of available cores.

orderedMerge	<i>Merge Two Data Frames Keeping the Order of the First</i>
--------------	---

Description

This is a simplified version of merge useful for merging additional data into a BGData object while keeping the order of the data in the BGData object.

Usage

```
orderedMerge(x, y, by = c(1L, 2L))
```

Arguments

x	Data frame
y	Data frame
by	Specifications of the columns used for merging. Defaults to the first two columns of the data frame, which traditionally has the family ID and the individual ID.

Value

Merged data frame

See Also

[BGData-class](#) for more information on the BGData class.

preprocess	<i>Center, scale, and impute data</i>
------------	---------------------------------------

Description

A faster version of [scale](#) with a similar interface that also allows for imputation. The main difference is that this version scales by the standard deviation regardless of whether centering is enabled or not. If centering is enabled, missing values are imputed by 0, otherwise by the mean of the column that contains the value.

Usage

```
preprocess(X, center = FALSE, scale = FALSE, impute = FALSE,
           nCores = getOption("mc.cores", 2L))
```

Arguments

X	A numeric matrix.
center	Either a logical value or numeric vector of length equal to the number of columns of X.
scale	Either a logical value or numeric vector of length equal to the number of columns of X.
impute	Indicates whether missing values should be imputed.
nCores	The number of cores (passed to mclapply). Defaults to the number of cores as detected by detectCores.

Value

The centered, scaled, and imputed matrix.

See Also

[scale](#), which this function tries to improve upon.

Examples

```
# Load example data
bg <- BGData:::loadExample()

# Center and scale genotypes
W <- preprocess(as.matrix(geno(bg)), center = TRUE, scale = TRUE)
```

readRAW

Creates a BGData Object From a .raw File or a .ped-Like File

Description

Creates a BGData object from a .raw file (generated with --recodeA in **PLINK**). Other text-based file formats are supported as well by tweaking some of the parameters as long as the records of individuals are in rows, and phenotypes, covariates and markers are in columns.

Usage

```
readRAW(fileIn, header = TRUE, dataType = integer(), n = NULL,
  p = NULL, sep = "", na.strings = "NA", nColSkip = 6L,
  idCol = c(1L, 2L), nNodes = NULL, linked.by = "rows",
  folderOut = paste0("BGData_", sub("\\.[:alnum:]]+$", "",
  basename(fileIn))), outputType = "byte", dimorder = if (linked.by ==
  "rows") 2L:1L else 1L:2L, verbose = FALSE)

readRAW_matrix(fileIn, header = TRUE, dataType = integer(), n = NULL,
  p = NULL, sep = "", na.strings = "NA", nColSkip = 6L,
```

```

idCol = c(1L, 2L), verbose = FALSE)

readRAW_big.matrix(fileIn, header = TRUE, dataType = integer(),
  n = NULL, p = NULL, sep = "", na.strings = "NA", nColSkip = 6L,
  idCol = c(1L, 2L), folderOut = paste0("BGData_",
  sub("\\.[:alnum:]]+$", "", basename(fileIn))), outputType = "char",
  verbose = FALSE)

```

Arguments

<code>fileIn</code>	The path to the plaintext file.
<code>header</code>	Whether <code>fileIn</code> contains a header. Defaults to TRUE.
<code>dataType</code>	The coding type of genotypes in <code>fileIn</code> . Use <code>integer()</code> or <code>double()</code> for numeric coding. Alpha-numeric coding is currently not supported for <code>readRAW</code> and <code>readRAW_big.matrix</code> : use the <code>--recodeA</code> option of PLINK to convert the <code>.ped</code> file into a <code>.raw</code> file. Defaults to <code>integer()</code> .
<code>n</code>	The number of individuals. Auto-detect if NULL. Defaults to NULL.
<code>p</code>	The number of markers. Auto-detect if NULL. Defaults to NULL.
<code>sep</code>	The field separator character. Values on each line of the file are separated by this character. If <code>sep = ""</code> (the default for <code>readRAW</code> the separator is "white space", that is one or more spaces, tabs, newlines or carriage returns.
<code>na.strings</code>	The character string used in the plaintext file to denote missing value. Defaults to NA.
<code>nColSkip</code>	The number of columns to be skipped to reach the genotype information in the file. Defaults to 6.
<code>idCol</code>	The index of the ID column. If more than one index is given, both columns will be concatenated with "_". Defaults to <code>c(1, 2)</code> , i.e. a concatenation of the first two columns.
<code>nNodes</code>	The number of nodes to create. Auto-detect if NULL. Defaults to NULL.
<code>linked.by</code>	If columns a column-linked matrix (<code>ColumnLinkedMatrix</code>) is created, if rows a row-linked matrix (<code>RowLinkedMatrix</code>). Defaults to rows.
<code>folderOut</code>	The path to the folder where to save the binary files. Defaults to the name of the input file (<code>fileIn</code>) without extension prefixed with "BGData_".
<code>outputType</code>	The <code>vmode</code> for <code>ff</code> and <code>type</code> for <code>big.matrix</code> objects. Default to <code>byte</code> for <code>ff</code> and <code>char</code> for <code>big.matrix</code> objects.
<code>dimorder</code>	The physical layout of the underlying <code>ff</code> object of each node.
<code>verbose</code>	Whether progress updates will be posted. Defaults to FALSE.

Details

The data included in the first couple of columns (up to `nColSkip`) is used to populate the sample information of a `BGData` object, and the remaining columns are used to fill the genotypes. If the first row contains a header (`header = TRUE`), data in this row is used to determine the column names for sample information and genotypes.

The genotypes can take several forms, depending on the function that is called (`readRAW`, `readRAW_matrix`, or `readRAW_big.matrix`). The following sections illustrate each function in detail.

readRAW

Genotypes are stored in a `LinkedMatrix` object where each node is an `ff` instance. Multiple `ff` files are used because the array size in `ff` is limited to the largest integer which can be represented on the system (`.Machine$integer.max`) and for genetic data this limitation is often exceeded. The `LinkedMatrix` package makes it possible to link several `ff` files together by columns or by rows and treat them similarly to a single matrix. By default a `ColumnLinkedMatrix` is used for the genotypes, but the user can modify this using the `linked.by` argument. The number of nodes to generate is either specified by the user using the `nNodes` argument or determined internally so that each `ff` object has a number of cells that is smaller than `.Machine$integer.max / 1.2`. A folder (see `folderOut`) that contains the binary flat files (named `geno_*.bin`) and an external representation of the `BGData` object in `BGData.RData` is created.

readRAW_matrix

Genotypes are stored in a regular `matrix` object. Therefore, this function will only work if the `.raw` file is small enough to fit into memory.

readRAW_big.matrix

Genotypes are stored in a filebacked `big.matrix` object. A folder (see `folderOut`) that contains the binary flat file (named `BGData.bin`), a descriptor file (named `BGData.desc`), and an external representation of the `BGData` object in `BGData.RData` are created.

Reloading a BGData object

To reload a `BGData` object, it is recommended to use the `load.BGData` function instead of the `load` function as `load` does not initialize `ff` objects or attach `big.matrix` objects.

See Also

[load.BGData\(\)](#) to load a previously saved `BGData` object, [as.BGData\(\)](#) to create `BGData` objects from non-text files (e.g. `.bed` files). [BGData-class](#), [ColumnLinkedMatrix-class](#), [RowLinkedMatrix-class](#), [big.matrix-class](#), and [ff](#) for more information on the above mentioned classes.

Examples

```
# Path to example data
path <- system.file("extdata", package = "BGData")

# Convert RAW files of chromosome 1 to a BGData object
bg <- readRAW(fileIn = paste0(path, "/chr1.raw"))

unlink("BGData_chr1", recursive = TRUE)
```

 segments

Find non-overlapping segments based on a summary statistic

Description

Given a summary statistic and a threshold, this function computes the number of non-overlapping segments, each defined as a discovery (i.e., $\text{statistic}[i] \leq \text{threshold}$) +/- a gap, in the same units as bp (often base-pair position).

Usage

```
segments(statistic, chr, bp, threshold, gap, trim = FALSE, verbose = FALSE)
```

Arguments

statistic	A statistic (e.g., BFDR or p-values).
chr	A vector containing the chromosome for each value of statistic.
bp	A vector containing the base-pair positions for each value of statistic.
threshold	The threshold to determine 'significance' (e.g., $1e^{-5}$ for p-values).
gap	1/2 of the length of the desired segments.
trim	Whether to collapse segments that were artificially inflated by gap. Defaults to FALSE.
verbose	Whether progress updates will be posted. Defaults to FALSE.

Value

A data frame containing the following information:

chr	Chromosome
start	Index where segment starts within statistic.
end	Index where segment ends within statistic.
length	Length of segment.
bpStart	Base-pair position where segment starts.
bpEnd	Base-pair position where segment ends.
bpLength	Length of segment in base-pair positions.
minValue	Smallest value of statistic within segment.
minValuePos	Position of variant with the smallest value of statistic within segment.

Examples

```

library(BGData)

# Load example data
bg <- BGData:::loadExample()

# Perform GWAS
pValues <- GWAS(
  formula = FT10 ~ 1,
  data = bg,
  method = "rayOLS"
)

# Determine segments within +/- 1MB from a significant variant
segments <- segments(
  statistic = pValues[, 4],
  chr = map(bg)$chromosome,
  bp = map(bg)$base_pair_position,
  threshold = 1e-5,
  gap = 1e6,
  trim = FALSE,
  verbose = FALSE
)

```

summarize

Generates Various Summary Statistics

Description

Computes the frequency of missing values, the (minor) allele frequency, and standard deviation of each column of X.

Usage

```

summarize(X, i = seq_len(nrow(X)), j = seq_len(ncol(X)),
  chunkSize = 5000L, nCores = getOption("mc.cores", 2L),
  verbose = FALSE)

```

Arguments

X	A matrix-like object, typically the genotypes of a BGData object.
i	Indicates which rows of X should be used. Can be integer, boolean, or character. By default, all rows are used.
j	Indicates which columns of X should be used. Can be integer, boolean, or character. By default, all columns are used.
chunkSize	The number of columns of X that are brought into physical memory for processing per core. If NULL, all elements in j are used. Defaults to 5000.

nCores	The number of cores (passed to mclapply). Defaults to the number of cores as detected by detectCores.
verbose	Whether progress updates will be posted. Defaults to FALSE.

Value

A data.frame with three columns: freq_na for frequencies of missing values, allele_freq for allele frequencies of the counted allele, and sd for standard deviations.

See Also

[file-backed-matrices](#) for more information on file-backed matrices. [multi-level-parallelism](#) for more information on multi-level parallelism. [BGData-class](#) for more information on the BGData class.

Examples

```
# Restrict number of cores to 1 on Windows
if (.Platform$OS.type == "windows") {
  options(mc.cores = 1)
}

# Load example data
bg <- BGData::loadExample()

# Summarize the whole dataset
sum1 <- summarize(X = geno(bg))

# Summarize the first 50 individuals
sum2 <- summarize(X = geno(bg), i = 1:50)

# Summarize the first 1000 markers (useful for distributed computing)
sum3 <- summarize(X = geno(bg), j = 1:1000)

# Summarize the first 50 individuals on the first 1000 markers
sum4 <- summarize(X = geno(bg), i = 1:50, j = 1:1000)

# Summarize by names
sum5 <- summarize(X = geno(bg), j = c("snp81233_C", "snp81234_C", "snp81235_T"))

# Convert to minor allele frequencies (useful if the counted alleles are not
# the minor alleles)
maf <- ifelse(sum1$allele_freq > 0.5, 1 - sum1$allele_freq, sum1$allele_freq)
```

Index

* **methods**

- geno, 11

- as.BGData, 3, 6
- as.BGData(), 22

- BGData, 4, 6
- BGData-class, 5
- BGData-package, 2

- chunkedApply, 6
- chunkedMap, 7

- detectCores, 19

- ff, 12, 18, 22
- file-backed-matrices, 8
- findRelated, 9
- fread, 4
- FWD, 10

- geno, 11
- geno, BGData-method (BGData-class), 5
- geno-class, 12
- geno<- (geno), 11
- geno<- ,BGData-method (BGData-class), 5
- getG, 12, 16
- getG_symDMatrix, 14
- glm, 17
- GWAS, 16

- lm, 17
- lm.fit, 17
- lmer, 17
- load.BGData, 6, 18
- load.BGData(), 22
- lsfit, 17

- map (geno), 11
- map, BGData-method (BGData-class), 5
- map<- (geno), 11
- map<- ,BGData-method (BGData-class), 5
- matrix, 12
- mclapply, 19
- multi-level-parallelism, 18

- orderedMerge, 19

- pheno (geno), 11
- pheno, BGData-method (BGData-class), 5
- pheno<- (geno), 11
- pheno<- ,BGData-method (BGData-class), 5
- preprocess, 19

- read.table, 4
- readRAW, 6, 20
- readRAW(), 4
- readRAW_big.matrix (readRAW), 20
- readRAW_matrix (readRAW), 20

- scale, 19, 20
- segments, 23
- SKAT, 17
- summarize, 24