Package 'BGData'

May 11, 2017

Version 1.0.0

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 (>= 3.0.0), BEDMatrix, LinkedMatrix, symDMatrix

Imports methods, parallel, bigmemory, ff, bit

Suggests data.table, lme4, SKAT, testthat

RoxygenNote 6.0.1

NeedsCompilation no

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

Maintainer Alexander Grueneberg <alexander.grueneberg@googlemail.com>

Repository CRAN

Date/Publication 2017-05-11 12:55:30 UTC

R topics documented:

`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:

- Memory mapping: 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 memory-mapped array may not be enough or convenient. A linked array is an array whose content is distributed over multiple memory-mapped 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 data structure called `BGData` that holds genotypes in the `@geno` slot, phenotypes in the `@pheno` slot, and additional information in the `@map` slot.

Memory-mapping

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

Multi-level parallelism

Functions with the `nCores`, `nTasks`, `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, and `nTasks` determines into how many tasks the problem is divided into. `nTasks` should be at least as high as `nCores` to keep all cores busy. Memory usage can be an issue for higher values of `nCores` and `nTasks` as R is not particularly memory-efficient. As a rule of thumb, at least around `object_size(X) + (nCores * (objec` MB of total memory will be needed for operations on memory-mapped matrices, not including potential copies of your data that might be created (for example `stats::lsfit()` runs `cbind(1, X)`). `i` and `j` can be used to include or exclude certain rows or columns. Internally, the `parallel::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` and `nTasks` to `1` as nodes are often cheaper than cores.

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 phenotypefile. 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::BEDMatrix-package, LinkedMatrix::LinkedMatrix-package, and symDMatrix::symDMatrix-package for an introduction to the respective packages.

`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 `@pheno` and `@map` slot, respectively. Currently supported are BEDMatrix::BEDMatrix objects, plain or nested in LinkedMatrix::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

x	An object. Currently supported are BEDMatrix::BEDMatrix objects, plain or nested in LinkedMatrix::ColumnLinkedMatrix objects.
alternatePhenotypeFile	Path to an alternate phenotypefile.
...	Additional arguments to the `utils::read.table()` or `data.table::fread()` call (if data.table 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 alternatephenotype file. The path to such a file can be provided with `alternatePhenotypeFile` and will be merged with the data in the `@pheno` slot.

For BEDMatrix::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 `@pheno` slot 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` will be generated. The same will happen for a .bim file for the `@map` slot.

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

Value

A BGData object.

See Also

`readRAW()` to convert text files to BGData objects.

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-class *An S4 Class to Represent Phenotype and Genotype Data.*

Description

This class is inspired by the phenotype/genotype file format .raw and its binary companion (also known as .bed) 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 one of the constructors `[BGData(...)][initialize,BGData-method]` or `[new("BGData", ...)][initialize,BGData-method]`.
- from a BED file using `as.BGData()`.
- from a previously saved BGData object using `load.BGData()`.
- from multiple files (even a mixture of different file types) using LinkedMatrix::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.

Slots

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

`pheno` A `data.frame` that contains phenotypes.

`map` A `data.frame` that contains a genetic map.

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

---

chunkedApply *Applies a Function on Each Row or Column of a Memory-Mapped Matrix-Like Object.*

Description

Similar to base::apply(), but designed for memory-mapped matrix-like objects. 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 base::apply(). If nTasks is greater than 1, the function will be applied in parallel using parallel::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)),
  bufferSize = 5000L, nTasks = nCores, nCores = getOption("mc.cores", 2L),
  verbose = FALSE, ...)
```

Arguments

X	A memory-mapped matrix-like object, typically @geno of a BGData object.
MARGIN	The subscripts which the function will be applied over. 1 indicates rows, 2 indicates columns.
FUN	The function to be applied.
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.
bufferSize	The number of rows or columns of X that are brought into RAM for processing. If NULL, all elements in i or j are used. Defaults to 5000.
nTasks	The number of tasks the problem should be broken into to be distributed among nCores cores. Defaults to nCores.
nCores	The number of cores (passed to parallel::mclapply()). Defaults to the number of cores as detected by parallel::detectCores().
verbose	Whether progress updates will be posted. Defaults to FALSE.
...	Additional arguments to be passed to the base::apply() like function.

Memory-mapping

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

Multi-level parallelism

Functions with the `nCores`, `nTasks`, `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, and `nTasks` determines into how many tasks the problem is divided into. `nTasks` should be at least as high as `nCores` to keep all cores busy. Memory usage can be an issue for higher values of `nCores` and `nTasks` as R is not particularly memory-efficient. As a rule of thumb, at least around `object_size(X) + (nCores * (objec` MB of total memory will be needed for operations on memory-mapped matrices, not including potential copies of your data that might be created (for example `stats::lsfit()` runs `cbind(1, X)`). `i` and `j` can be used to include or exclude certain rows or columns. Internally, the `parallel::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` and `nTasks` to 1 as nodes are often cheaper than cores.

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 = bg@geno, MARGIN = 2, FUN = sd)
```

---

crossprod_parallel *Computes crossprod (x'x or x'y) or tcrossprod (xx' or xy') in Parallel.*

Description

Similar to `base::crossprod()` and `base::tcrossprod()`, but designed to carry out operations in parallel. The input matrix `x` (and `y` if not `NULL`) is broken into `nTasks` chunks and passed to `parallel::mclapply()` which performs `base::crossprod()` or `base::tcrossprod()` on each chunk. The results are added up and returned.

Usage

```
crossprod_parallel(x, y = NULL, nTasks = nCores,
  nCores = getOption("mc.cores", 2L))

tcrossprod_parallel(x, y = NULL, nTasks = nCores,
  nCores = getOption("mc.cores", 2L))
```

Arguments

x	A matrix-like object, typically @geno of a BGData object.
y	vector or matrix-like object. NULL by default.
nTasks	The number of tasks the problem should be broken into to be distributed among nCores cores. Defaults to nCores.
nCores	The number of cores (passed to parallel::mclapply()). Defaults to the number of cores as detected by parallel::detectCores().

Details

If nTasks is 1, base::crossprod() or base::tcrossprod() will be called directly without parallelism.

Value

x'x or x'y (crossprod_parallel), or xx' or xy' (tcrossprod_parallel), depending on whether y is provided.

Multi-level parallelism

Functions with the nCores, nTasks, 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, and nTasks determines into how many tasks the problem is divided into. nTasks should be at least as high as nCores to keep all cores busy. Memory usage can be an issue for higher values of nCores and nTasks as R is not particularly memory-efficient. As a rule of thumb, at least around object_size(X) + (nCores * (objec MB of total memory will be needed for operations on memory-mapped matrices, not including potential copies of your data that might be created (for example stats::lsfit() runs cbind(1, X)). i and j can be used to include or exclude certain rows or columns. Internally, the parallel::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 and nTasks to 1 as nodes are often cheaper than cores.

See Also

getG() to compute a genomic relationship matrix.

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 xx' in parallel
tcrossprod_parallel(x = bg@geno)

# Compute xy' in parallel (see getG)
tcrossprod_parallel(x = bg@geno, y = bg@geno[1:50, ])

# Compute x'x in parallel
crossprod_parallel(x = bg@geno)
```

`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::LinkedMatrix, BEDMatrix::BEDMatrix, bigmemory::big.matrix, `ff_matrix`, and `matrix`.

See Also

The `@geno` slot of BGData 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, scaleG = TRUE, minVar = 1e-05,
  i = seq_len(nrow(X)), j = seq_len(ncol(X)), i2 = NULL,
  bufferSize = 5000L, nTasks = nCores, nCores = getOption("mc.cores", 2L),
  verbose = FALSE)
```

Arguments

X	A matrix-like object, typically @geno 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.
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.
bufferSize	The number of columns of X that are brought into RAM for processing. If NULL, all columns of X are used. Defaults to 5000.
nTasks	The number of tasks the problem should be broken into to be distributed among nCores cores. Defaults to nCores.
nCores	The number of cores (passed to parallel::mclapply()). Defaults to the number of cores as detected by parallel::detectCores().
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 base::tcrossprod().

Value

A positive semi-definite symmetric numeric matrix.

Memory-mapping

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

Multi-level parallelism

Functions with the nCores, nTasks, 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, and nTasks determines into how many tasks the problem is divided into. nTasks should be at least as high as nCores to keep all cores busy. Memory usage can be an issue for higher values of nCores and nTasks as R is not particularly memory-efficient. As a rule of thumb, at least around object_size(X) + (nCores * (objec MB of total memory will be needed for operations on memory-mapped matrices, not including potential copies of your data that might be created (for example stats::lsfit() runs cbind(1, X)). i and j can be used to include or exclude certain rows or columns. Internally, the parallel::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 and nTasks to 1 as nodes are often cheaper than cores.

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 = bg@geno)

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

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

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

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

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

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

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

# Compute unscaled G matrix by combining blocks of $XX_{i2}'$ where $X_{i2}$ 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 = bg@geno, i2 = 1:100, center = centers, scale = scales)
block2 <- getG(X = bg@geno, i2 = 101:199, center = centers, scale = scales)
g8 <- cbind(block1, block2)

# Compute unscaled G matrix by combining blocks of $X_{i}X_{i2}'$ where both
# $X_{i}$ 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 = bg@geno, i = 1:100, i2 = 1:100, center = centers, scale = scales)
block12 <- getG(X = bg@geno, i = 1:100, i2 = 101:199, center = centers, scale = scales)
block21 <- getG(X = bg@geno, i = 101:199, i2 = 1:100, center = centers, scale = scales)
block22 <- getG(X = bg@geno, 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, scaleG = TRUE,
  folderOut = paste0("symDMatrix_", randomString()), vmode = "double",
  i = seq_len(nrow(X)), j = seq_len(ncol(X)), blockSize = 5000L,
  nTasks = nCores, nCores = getOption("mc.cores", 2L), verbose = FALSE)
```

Arguments

`X`	A matrix-like object, typically `@geno` of a BGData object.
`center`	Either a logical value or a numeric vector of length equal to the number of columns of `X`. 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. If FALSE, no scaling is done. Defaults to TRUE.
scaleG	TRUE/FALSE whether xx' must be scaled.
folderOut	The path to the folder where to save the symDMatrix::symDMatrix object. Defaults to a random string prefixed with "symDMatrix_".
vmode	vmode of ff objects.
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.
blockSize	The number of rows and columns of each block. If NULL, a single block of the same length as i will be created. Defaults to 5000.
nTasks	The number of tasks the problem should be broken into to be distributed among nCores cores. Defaults to nCores.
nCores	The number of cores (passed to parallel::mclapply()). Defaults to the number of cores as detected by parallel::detectCores().
verbose	Whether progress updates will be posted. Defaults to FALSE.

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::symDMatrix object.

Multi-level parallelism

Functions with the nCores, nTasks, 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, and nTasks determines into how many tasks the problem is divided into. nTasks should be at least as high as nCores to keep all cores busy. Memory usage can be an issue for higher values of nCores and nTasks as R is not particularly memory-efficient. As a rule of thumb, at least around object_size(X) + (nCores * (objec MB of total memory will be needed for operations on memory-mapped matrices, not including potential copies of your data that might be created (for example stats::lsfit() runs cbind(1, X)). i and j can be used to include or exclude certain rows or columns. Internally, the parallel::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 and nTasks to 1 as nodes are often cheaper than cores.

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 `@geno` 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(data@geno)),
  j = seq_len(ncol(data@geno)), bufferSize = 5000L, nTasks = nCores,
  nCores = getOption("mc.cores", 2L), verbose = FALSE, ...)
```

Arguments

`formula`	The formula for the GWAS model without including the marker, e.g. `y ~ 1` or `y ~ factor(sex) + age`. The variables included in the formula must be in the `@pheno` object of the BGData.
`data`	A BGData object.
`method`	The regression method to be used. Currently, the following methods are implemented: `stats::lm()`, `stats::lm.fit()`, `stats::lsfit()`, `stats::glm()`, `lme4::lmer()`, and `SKAT::SKAT()`. Defaults to `lsfit`.
`i`	Indicates which rows of `@geno` should be used. Can be integer, boolean, or character. By default, all rows are used.
`j`	Indicates which columns of `@geno` should be used. Can be integer, boolean, or character. By default, all columns are used.
`bufferSize`	The number of columns of `@geno` that are brought into RAM for processing. If `NULL`, all elements in `j` are used. Defaults to 5000.
`nTasks`	The number of tasks the problem should be broken into to be distributed among `nCores` cores. Defaults to `nCores`.
`nCores`	The number of cores (passed to `parallel::mclapply()`). Defaults to the number of cores as detected by `parallel::detectCores()`.
`verbose`	Whether progress updates will be posted. Defaults to `FALSE`.
`...`	Additional arguments for chunkedApply and regression method.

Value

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

Memory-mapping

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

Multi-level parallelism

Functions with the `nCores`, `nTasks`, `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, and `nTasks` determines into how many tasks the problem is divided into. `nTasks` should be at least as high as `nCores` to keep all cores busy. Memory usage can be an issue for higher values of `nCores` and `nTasks` as R is not particularly memory-efficient. As a rule of thumb, at least around `object_size(X) + (nCores * (objec` MB of total memory will be needed for operations on memory-mapped matrices, not including potential copies of your data that might be created (for example `stats::lsfit()` runs `cbind(1, X)`). `i` and `j` can be used to include or exclude certain rows or columns. Internally, the `parallel::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` and `nTasks` to 1 as nodes are often cheaper than cores.

Examples

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

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

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

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

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

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

```
# Perform a single marker regression on the first 50 markers (useful for
# distributed computing)
res4 <- GWAS(formula = FT10 ~ 1, data = bg, j = 1:50)
```

`initialize,BGData-method` *Creates a New BGData Instance.*

Description

This method is run when a BGData object is created using `BGData(...)` or `new("BGData", ...)`.

Usage

```
## S4 method for signature 'BGData'
initialize(.Object, geno, pheno, map)
```

Arguments

`.Object`	The BGData instance to be initialized. This argument is passed in by R and can be ignored, but still needs to be documented.
`geno`	A geno object that contains genotypes. geno is a class union of several matrix-like types, many of them suitable for very large datasets. Currently supported are LinkedMatrix::LinkedMatrix, BEDMatrix::BEDMatrix, bigmemory::big.matrix, `ff_matrix`, and `matrix`.
`pheno`	A `data.frame` that contains phenotypes. A stub that only contains an `IID` column populated with the rownames of `@geno` will be generated if missing.
`map`	A `data.frame` that contains a genetic map. A stub that only contains a `mrk` column populated with the colnames of `@geno` will be generated if missing.

`load.BGData` *Loads BGData (and Other) Objects from .RData Files.*

Description

This function is similar to `base::load()`, but also initializes the different types of objects that the `@geno` slot of a BGData object can take. Currently supported are `ff_matrix`, bigmemory::big.matrix, and BEDMatrix::BEDMatrix objects. If the object is of type LinkedMatrix::LinkedMatrix, all nodes will be initialized with their appropriate method.

Usage

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

Arguments

`file`	The name of the .RData file to be loaded.
`envir`	The environment where to load the data.

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

`fileIn`	The path to the plaintext file.
`header`	Whether `fileIn` contains a header. Defaults to `TRUE`.
`dataType`	The coding type of genotypes in `fileIn`. Use `integer()` or `double()` for numeric coding. Alpha-numeric coding is currently not supported for `readRAW()` and `readRAW_big.matrix()`: use the `--recodeA` option of PLINK to convert the .ped file into a .raw file. Defaults to `integer()`.
`n`	The number of individuals. Auto-detect if `NULL`. Defaults to `NULL`.
`p`	The number of markers. Auto-detect if `NULL`. Defaults to `NULL`.
`sep`	The field separator character. Values on each line of the file are separated by this character. If `sep = ""` (the default for `readRAW()` the separator is "white space", that is one or more spaces, tabs, newlines or carriage returns.
`na.strings`	The character string used in the plaintext file to denote missing value. Defaults to `NA`.
`nColSkip`	The number of columns to be skipped to reach the genotype information in the file. Defaults to `6`.

idCol	The index of the ID column. If more than one index is given, both columns will be concatenated with "_". Defaults to c(1, 2), i.e. a concatenation of the first two columns.
nNodes	The number of nodes to create. Auto-detect if NULL. Defaults to NULL.
linked.by	If columns a column-linked matrix (LinkedMatrix::ColumnLinkedMatrix) is created, if rows a row-linked matrix (LinkedMatrix::RowLinkedMatrix). Defaults to rows.
folderOut	The path to the folder where to save the binary files. Defaults to the name of the input file (fileIn) without extension prefixed with "BGData_".
outputType	The vmode for ff and type for bigmemory::big.matrix) objects. Default to byte for ff and char for bigmemory::big.matrix objects.
dimorder	The physical layout of the underlying ff object of each node.
verbose	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 @pheno slot of a BGData object, and the remaining columns are used to fill the @geno slot. If the first row contains a header (header = TRUE), data in this row is used to determine the column names for @pheno and @geno.

@geno 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::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$integer.max) and for genetic data this limitation is often exceeded. The LinkedMatrix::LinkedMatrix package makes it possible to link several ff files together by columns or by rows and treat them similarly to a single matrix. By default a LinkedMatrix::ColumnLinkedMatrix is used for @geno, but the user can modify this using the linked.by argument. The number of nodes to generate is either specified by the user using the nNodes argument or determined internally so that each ff object has a number of cells that is smaller than .Machine$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 bigmemory::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 `base::load()` function as `base::load()` does not initialize `ff` objects or attach bigmemory::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).

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"))
```

---

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)),
  bufferSize = 5000L, nTasks = nCores, nCores = getOption("mc.cores", 2L),
  verbose = FALSE)
```

Arguments

`X`	A matrix-like object, typically `@geno` 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.
`bufferSize`	The number of columns of `X` that are brought into RAM for processing. If `NULL`, all elements in `j` are used. Defaults to 5000.
`nTasks`	The number of tasks the problem should be broken into to be distributed among `nCores` cores. Defaults to `nCores`.
`nCores`	The number of cores (passed to `parallel::mclapply()`). Defaults to the number of cores as detected by `parallel::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 (minor) allele frequencies, and `sd` for standard deviations.

Memory-mapping

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

Multi-level parallelism

Functions with the `nCores`, `nTasks`, `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, and `nTasks` determines into how many tasks the problem is divided into. `nTasks` should be at least as high as `nCores` to keep all cores busy. Memory usage can be an issue for higher values of `nCores` and `nTasks` as R is not particularly memory-efficient. As a rule of thumb, at least around `object_size(X) + (nCores * (objec` MB of total memory will be needed for operations on memory-mapped matrices, not including potential copies of your data that might be created (for example `stats::lsfit()` runs `cbind(1, X)`). `i` and `j` can be used to include or exclude certain rows or columns. Internally, the `parallel::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` and `nTasks` to 1 as nodes are often cheaper than cores.

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 = bg@geno)

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

# Summarize the first 1000 markers (useful for distributed computing)
sum3 <- summarize(X = bg@geno, j = 1:100)
```

```
# Summarize the first 50 individuals on the first 1000 markers
sum4 <- summarize(X = bg@geno, i = 1:50, j = 1:100)

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

Index