Package ‘BigDataStatMeth’

March 30, 2022

Type Package

Title Statistical Methods and Algorithms for Big Data

Version 0.99.32

Description Basic Algebra methods using parallel algorithms to be used in big data problems such as omics data analyses. The functions will consider as input an HDF5 data file, an object of class DelayedArray or an R object.

License MIT + file LICENSE

Depends R (>= 3.6.0)

Imports data.table, Rcpp (>= 1.0.6), RcppParallel (>= 5.0.2), RCurl, rhdf5, utils

LinkingTo Rcpp, RcppEigen, RcppParallel, beachmat, Rhdf5lib, RSpectra, BH

Suggests HDF5Array, DelayedArray, Matrix, BiocStyle, knitr, rmarkdown, ggplot2, microbenchmark

SystemRequirements GNU make, C++11

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.1.2

NeedsCompilation yes

Author Dolors Pelegri-Siso [aut, cre]

Juan R. Gonzalez [aut] (<https://orcid.org/0000-0003-3267-2146>)

Maintainer Dolors Pelegri-Siso <dolors.pelegri@isglobal.org>

Repository CRAN

Date/Publication 2022-03-29 22:00:07 UTC
R topics documented:

bdAdd_hdf5_matrix ........................................ 3
bdapply_Function_hdf5 ................................ 4
bdBind_hdf5 ............................................. 5
bdblockmult ............................................. 6
bdblockmult_hdf5 ....................................... 7
bdblockmult_sparse ..................................... 9
bdblockmult_sparse_hdf5 ................................ 10
bdblockmult_vector .................................... 11
bdCCA_hdf5 ............................................. 12
bdCreateEmptyDataset_hdf5 ............................. 13
bdCreateGroup_hdf5 ................................... 14
bdCreateLink_hdf5 ...................................... 15
bdCreate_hdf5_matrix_file .............................. 16
bdCrossprod ........................................... 17
bdCrossprod_hdf5 .................................... 18
bdCrossprod_Weighted ................................ 19
bdctrsm ................................................. 20
bdgetDatasetsList_hdf5 ................................ 21
bdgetDim_hdf5 ......................................... 21
bdget_maf_hdf5 ......................................... 22
bdImportData_hdf5 ..................................... 23
bdImport_text_to_hdf5 .................................. 24
bdImpute_snps_hdf5 .................................... 25
bdInvCholesky .......................................... 26
bdlm_paral ............................................. 27
bdNormalize_Data ...................................... 28
bdNormalize_hdf5 ..................................... 29
bdparallelpow2 ........................................ 30
bdparallelVectorSum .................................. 30
bdPCA_hdf5 ............................................ 31
bdpseudoinv ........................................... 32
bdQR ...................................................... 32
bdReduce_matrix_hdf5 .................................. 33
bdRemovelowdata ...................................... 34
bdRemove_hdf5_element .................................. 34
bdremove_maf_hdf5 ..................................... 35
bdScalarwproduct ...................................... 36
bdSolve ................................................ 37
bdSplit_matrix_hdf5 ................................... 38
bdSVD ................................................... 39
bdSVD_hdf5 ............................................ 40
bdSVD_lapack .......................................... 41
bdtCrossprod .......................................... 42
bdtCrossprod_hdf5 .................................... 43
bdtCrossprod_Weighted ................................ 45
bdwproduct ............................................. 46
bdAdd_hdf5_matrix

Write matrix to existing hdf5 file

Description

Creates a hdf5 file with numerical data matrix,

Usage

bdAdd_hdf5_matrix(
  object,
  filename,
  group,
  dataset,
  transp = NULL,
  force = FALSE
)

Arguments

object numerical data matrix
filename character array indicating the name of the file to create
group character array indicating folder or group name to put the matrix in hdf5 file
dataset character array indicating the dataset name that contains the matrix data
transp boolean if true, data is manipulated in transposed form
force optional boolean if true and file exists, removes old file and creates a new file

Value

none
bdapply_Function_hdf5

Examples

```r
matA <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15), nrow = 3, byrow = TRUE)
matB <- matrix(c(15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,5,3,4,5,2,6,2,3,4,
                 42, 23, 23, 423,1,2), nrow = 3, byrow = TRUE)
bdCreate_hdf5_matrix_file("BasicMatVect.hdf5", matA, "INPUT", "matA")
bdAdd_hdf5_matrix(matB, "BasicMatVect.hdf5", "INPUT", "matB")

# Remove file (used as example)
if (file.exists("BasicMatVect.hdf5")) {
  # Delete file if it exist
  file.remove("BasicMatVect.hdf5")
}
```

bdapply_Function_hdf5

Apply function to different datasets inside a group

Description

Apply function to different datasets inside a group

Usage

```r
bdapply_Function_hdf5(
  filename, 
  group, 
  datasets, 
  outgroup, 
  func, 
  b_group = NULL, 
  b_datasets = NULL, 
  force = FALSE 
)
```

Arguments

- `filename`: Character array, indicating the name of the file to create
- `group`: Character array, indicating the input group where the data set to be imputed is.
- `datasets`: Character array, indicating the input datasets to be used
- `outgroup`: Character array, indicating group where the data set will be saved after imputation if `outgroup` is NULL, output dataset is stored in the same input group.
- `func`: Character array, function to be applied: QR to apply bdQR() function to datasets
- `CrossProd` to apply bdCrossprod() function to datasets
- `tCrossProd` to apply bdtCrossprod() function to datasets
- `invChol` to apply bdInvCholesky() function to datasets
- `blockmult` to apply matrix multiplication, in that case, we need the datasets to be
bdBind_hdf5

used defined in b_datasets variable, datasets and b_datasets must be of the same length, in that case, the operation is performed according to index, for example, if we have datasets = "A1", "A2", "A3" and b_datasets = "B1", "B2", "B3", the functions performs: A1 CrossProd_double to performs crossprod using two matrices, see blockmult tCrossProd_double to performs transposed crossprod using two matrices, see blockmult solve to solve matrix equation system, see blockmult for parametrization

b_group, optional Character array indicating the input group where data are stored when we need a second dataset to operate, for example in functions like matrix multiplication

b_datasets, optional Character array indicating the input datasets to be used when we need a second dataset in functions like matrix multiplication

force, optional Boolean if true, previous results in same location inside hdf5 will be overwritten, by default force = false, data was not overwritten.

Value

Original hdf5 data file with results after apply function to different datasets

Original hdf5 data file with results after apply function to different datasets

bdBind_hdf5

Bind matrices by rows or columns

Description

Merge existing matrices inside hdf5 data file by rows or by column

Usage

bdBind_hdf5(
  filename,
  group,
  datasets,
  outgroup,
  outdataset,
  func,
  force = FALSE
)

Arguments

filename, character array indicating the name of the file to create

group, character array indicating the input group where the data set to be imputed is.

datasets, character array indicating the input dataset to be imputed

outgroup, character array indicating group where the data set will be saved after imputation if ‘outgroup’ is NULL, output dataset is stored in the same input group.
outdataset, character array indicating the name for the new merged dataset
func, character array function to be applied
bindRows merge datasets by rows
bindCols apply datasets by columns
force, boolean if true, previous results in same location inside hdf5 will be overwritten.

Value
Original hdf5 data file with results after input datasets

bdbblockmult

Block matrix multiplication with Delayed Array Object

Description
This function performs a block matrix-matrix multiplication with numeric matrix or Delayed Arrays

Usage
bdbblockmult(
  A,
  B,
  block_size = 128,
  paral = TRUE,
  threads = NULL,
  bigmatrix = 10000,
  mixblock_size = 128,
  outfile = "tmp_blockmult.hdf5",
  onmemory = FALSE
)

Arguments
A a double matrix.
B a double matrix.
block_size (optional, default = 128) block size to make matrix multiplication, if ‘block_size = 1’ no block size is applied (size 1 = 1 element per block)
paral, (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation
threads (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available
bigmatrix (optiona, default = 5000) maximum number of rows or columns to consider as big matrix and work with hdf5 files, by default a matrix with more than 5000 rows or files is considered big matrix and computation is made in disk
mixblock_size (optional, default = 128), only if we are working with big matrix and parallel computation = true. Block size for mixed computation in big matrix parallel. Size of the block to be used to perform parallelized memory memory of the block read from the disk being processed.

outfile (optional) file name to work with hdf5 if we are working with big matrix in disk.

onmemory (optional) if onmemory = TRUE the multiplication is forced to execute in memory.

Value

numerical matrix

Examples

# with numeric matrix

m <- 500
k <- 1500
n <- 400
A <- matrix(rnorm(n*k), nrow=n, ncol=k)
B <- matrix(rnorm(n*k), nrow=k, ncol=n)

C <- bdblockmult(A, B, 128, TRUE)

bdblockmult_hdf5

Block matrix multiplication with hdf5 datasets

Description

This function performs a block matrix-matrix multiplication with matrix stored in HDF5 file

Usage

bdblockmult_hdf5(
    filename,
    group,
    a,
    b,
    groupB = NULL,
    block_size = 128,
    paral = FALSE,
    threads = NULL,
    mixblock_size = 128,
    outgroup = "OUTPUT",
    outdataset = NULL
)
**Arguments**

- **filename** string file name where dataset to normalize is stored
- **group** string with the group name where matrix is stored inside HDF5 file
- **a** a double matrix.
- **b** a double matrix.
- **groupB,** string, (optional) group name where dataset b is stored
- **block_size** (optional, default = 128) block size to make matrix multiplication, if ‘block_size = 1’ no block size is applied (size 1 = 1 element per block)
- **paral,** (optional, default = FALSE) paral = true –> TO BE IMPLEMENTED
- **threads** (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available
- **mixblock_size** (optiona, default = 128), only if we are working with big matrix and parallel computation = true. Block size for mixed computation in big matrix parallel. Size of the block to be used to perform parallelized memory memory of the block read from the disk being processed.
- **outgroup** (optional) string with group name where we want to store the result matrix
- **outdataset** (optional) string with dataset name where we want to store the results

**Value**

A list with an HDF5 object with numerical matrix and HDF5 file name with results

```r
##
• "res"rhdf5 object with result matrix - link to hdf5 file contents . IMPORTANT !!, we have to close the object after finish to work
• "file"string with hdf5 file name with result and original matrixs
• "dataset"string complete path inside hdf5 file where results are stored.
```

**Examples**

```r
# with numeric matrix

m <- 500
k <- 1500
n <- 400
A <- matrix(rnorm(n*k), nrow=n, ncol=k)
B <- matrix(rnorm(n*k), nrow=k, ncol=n)
C <- bdblockmult(A, B, 128, TRUE)
```
**bdblockmult_sparse**  
Block matrix multiplication

**Description**
This function performs a block matrix-matrix multiplication with numeric matrix

**Usage**

```
bdblockmult_sparse(A, B, paral = NULL, threads = NULL)
```

**Arguments**
- **A**: a sparse double matrix.
- **B**: a sparse double matrix.
- **paral**, (optional, default = TRUE) if `paral = TRUE` performs parallel computation else performs seria computation
- **threads** (optional) only if `paral = true`, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available

**Value**
Sparse matrix with results

**Examples**

```r
library(Matrix)
library(BigDataStatMeth)

k <- 1e3
set.seed(1)
x_sparse <- sparseMatrix(i = sample(x = k, size = k),
  j = sample(x = k, size = k),
  x = rnorm(n = k))
set.seed(2)
y_sparse <- sparseMatrix(i = sample(x = k, size = k),
  j = sample(x = k, size = k),
  x = rnorm(n = k))

d <- bdblockmult_sparse(x_sparse, y_sparse)
```
**bdblockmult_sparse_hdf5**

*Block matrix multiplication*

**Description**

This function performs a block matrix-matrix multiplication with numeric matrix

**Usage**

```r
bdblockmult_sparse_hdf5(filename, group, A, B, outgroup = NULL)
```

**Arguments**

- **filename**: string file name where dataset to normalize is stored
- **group**: string Matrix
- **A**: string with dataset name where matrix is stored
- **B**: string with dataset name where matrix is stored
- **outgroup**: string with de group name under the matrix will be stored

**Value**

list with filename and the group and dataset name under the results are stored

**Examples**

```r
library(Matrix)
library(BigDataStatMeth)

k <- 1e3
set.seed(1)
x_sparse <- sparseMatrix(
  i = sample(x = k, size = k),
  j = sample(x = k, size = k),
  x = rnorm(n = k)
)
set.seed(2)
y_sparse <- sparseMatrix(
  i = sample(x = k, size = k),
  j = sample(x = k, size = k),
  x = rnorm(n = k)
)

if( isTRUE(file.exists("BasicMatVect.hdf5"))) {
  file.remove("BasicMatVect.hdf5")
}
bdCreate_hdf5_matrix_file("BasicMatVect.hdf5", as.matrix(x_sparse), "SPARSE", "x_sparse")
```
bdAdd_hdf5_matrix(as.matrix(y_sparse), "BasicMatVect.hdf5", "SPARSE", "y_sparse")

d <- bdblockmult_sparse_hdf5("BasicMatVect.hdf5", "SPARSE", "x_sparse", "y_sparse")

# Remove file (used as example)
if (file.exists("BasicMatVect.hdf5")) {
  # Delete file if it exist
  file.remove("BasicMatVect.hdf5")
}


### bdblockmult_vector

**Block matrix-vector multiplication with Delayed Array Object**

**Description**

This function performs a block matrix-vector multiplication with R-Objects or Delayed Arrays.

**Usage**

```r
bdblockmult_vector(
  A,
  b,
  block_size = 128,
  paral = TRUE,
  threads = NULL,
  outfile = "tmp_blockmult_vector.hdf5",
  onmemory = FALSE
)
```

**Arguments**

- **A**: a double matrix.
- **b**: a double vector or array.
- **block_size**: (optional, default = 128) block size to make matrix multiplication, if ‘block_size = 1’ no block size is applied (size 1 = 1 element per block)
- **paral**: (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation
- **threads**: (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available
- **outfile**: (optional) file name to work with hdf5 if we are working with big matrix in disk.
- **onmemory**: (optional) if onmemory = TRUE the multiplication is forced to execute in memory

**Value**

numerical matrix
Examples

# with numeric matrix

k <- 100
n <- 400
A <- matrix(rnorm(n*k), nrow=n, ncol=k)
B <- sample(1:100,100, replace = TRUE);

res <- bdblockmult_vector(A, B, 128, TRUE)

---

bdCCA_hdf5  

**Canonical Correlation Analysis**

Description

This function is an application of the BigDataStatMeth functions to generate new methods. This function perform a Canonical Correlation Analysis from two matrices stored in hdf5 data file. This function applies matrix partitioning, merge bloks to create a full matrix, apply a function to different blocks, etc.

Usage

bdCCA_hdf5(

    filename,
    X,
    Y,
    m = 10,
    bcenter = TRUE,
    bscale = FALSE,
    bycols = FALSE,
    overwriteResults = FALSE,
    keepInteResults = FALSE,
    threads = 1,
    k = 4,
    q = 1

)

Arguments

- **filename**  
  string file name where dataset to normalize is stored.

- **X**  
  Dataset, path inside the hdf5 data file.

- **Y**  
  Dataset, path inside the hdf5 data file.

- **m**  
  Integer, number of blocks in which we want to partition the matrix to perform the calculations.
bdCreateEmptyDataset_hdf5

bcenter, Boolean, if true, dataset is centered to perform calculus.
bscale, Boolean, if true, dataset is centered to perform calculus.
bycols, Boolean by default = true, true indicates that the imputation will be done by columns, otherwise, the imputation will be done by rows.
overwriteResults, Boolean, if true, datasets existing inside a file must be overwritten if we are using the same names.
keepInteResults, Boolean, if false, intermediate results will be removed.
threads (optional) only used in some operations inside function. If threads is null then threads = maximum number of threads available - 1.
k (optional) number of local SVDs to concatenate at each level
q (optional) number of levels

Value

hdf5 data file with CCA results,

Examples

print("Example in vignette")
bdCreateGroup_hdf5

Arguments

- filename, character array indicating the name of the file to create
- group, string with name of the group where the new dataset will be created
- dataset, string with name for the new dataset
- nrows, integer with the number of rows for the new dataset
- ncols, integer with the number of columns for the new dataset
- overwrite, optional boolean if true datasets exists, replaces old dataset with a new empty dataset

Value

none

Examples

```r
matA <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15), nrow = 3, byrow = TRUE)
bdCreate_hdf5_matrix_file("BasicMatVect.hdf5", matA, "INPUT", "matA")
bdCreateEmptyDataset_hdf5("BasicMatVect.hdf5", "INPUT", "EmptyMat", 100, 10)
```

# Remove file (used as example)
if (file.exists("BasicMatVect.hdf5")) {
  # Delete file if it exist
  file.remove("BasicMatVect.hdf5")
}

---

bdCreateGroup_hdf5  Create groups

Description

Create groups in hdf5 data file

Usage

```r
bdCreateGroup_hdf5(filename, group)
```

Arguments

- filename, character array indicating the name of the file to create
- group, string with the name for the new group (complete route)
bdCreateLink_hdf5

Value
none

Examples

matA <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15), nrow = 3, byrow = TRUE)

bdCreate_hdf5_matrix_file("BasicMatVect.hdf5", matA, "INPUT", "matA", force = TRUE)
bdCreateGroup_hdf5("BasicMatVect.hdf5", "INPUT/NEWGROUP")
bdCreateGroup_hdf5("BasicMatVect.hdf5", "NEWGROUP2")

# Remove file (used as example)
if (file.exists("BasicMatVect.hdf5")) {
  # Delete file if it exist
  file.remove("BasicMatVect.hdf5")
}

bdCreateLink_hdf5
Create hard link between two datasets

Description
Create hard link between two datasets

Usage
bdCreateLink_hdf5(filename, source, dest)

Arguments
filename, character array indicating the name of the file to create
source, string with route to source dataset
dest, string with route to destination dataset

Value
none
bdCreate_hdf5_matrix_file

Create hdf5 data file and write data to it

**Description**

Creates a hdf5 file with numerical data matrix.

**Usage**

```r
bdCreate_hdf5_matrix_file(
  filename,
  object,
  group = NULL,
  dataset = NULL,
  transp = NULL,
  force = NULL
)
```

**Arguments**

- `filename`, character array indicating the name of the file to create
- `object`, numerical data matrix
- `group`, character array indicating folder name to put the matrix in hdf5 file
- `dataset`, character array indicating the dataset name to store the matrix data
- `transp`, boolean, if trans=true matrix is stored transposed in hdf5 file
- `force`, optional boolean if true and file exists, removes old file and creates a new file with the dataset data.

**Value**

none

**Examples**

```r
matA <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15), nrow = 3, byrow = TRUE)
bdCreate_hdf5_matrix_file("BasicMatVect.hdf5", matA, "INPUT", "matA")
```

# Remove file (used as example)
if (file.exists("BasicMatVect.hdf5")) {
  # Delete file if it exist
  file.remove("BasicMatVect.hdf5")
}
bdCrossprod

Crossproduct

Description

This function performs a crossproduct or transposed crossproduct of numerical or DelayedArray matrix.

Usage

bdCrossprod(A, B = NULL, block_size = 256, paral = TRUE, threads = NULL)

Arguments

A  numerical or Delayed Array matrix
B  optional, numerical or Delayed Array matrix
block_size (optional, default = 128) block size to make matrix multiplication, if ‘block_size = 1’ no block size is applied (size 1 = 1 element per block)
paral, (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation
threads (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available

Value

numerical matrix with crossproduct

Examples

n <- 100
p <- 60
X <- matrix(rnorm(n*p), nrow=n, ncol=p)
res <- bdCrossprod(X)
all.equal(crossprod(X), res)

n <- 100
p <- 100
Y <- matrix(rnorm(n*p), nrow=n)
# With two matrices
res <- bdCrossprod(X,Y)
**bdCrossprod_hdf5**  
*Crossprod with hdf5 matrix*

**Description**

This function performs the crossprod from a matrix inside and hdf5 data file.

**Usage**

```r
bdCrossprod_hdf5(
  filename,  # string file name where dataset to normalize is stored
  group,    # string, group name where dataset A is stored
  A,        # string name inside HDF5 file
  groupB = NULL,  # string, group name where dataset b is stored
  B = NULL,  # string, dataset name for matrix B inside HDF5 file
  block_size = NULL,  # (optional, default = 128) block size to make matrix multiplication, if 'block_size = 1' no block size is applied (size 1 = 1 element per block)
  paral = NULL,  # (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation
  threads = NULL,  # (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available
  mixblock_size = NULL,  # (optional) only for debug purpose
  outgroup = NULL  # (optional) group name to store results from Crossprod inside hdf5 data file
)
```

**Arguments**

- `filename`: string file name where dataset to normalize is stored
- `group`: string, group name where dataset A is stored
- `A`: string name inside HDF5 file
- `groupB`: string, group name where dataset b is stored
- `B`: string, dataset name for matrix B inside HDF5 file
- `block_size`: (optional, default = 128) block size to make matrix multiplication, if ‘block_size = 1’ no block size is applied (size 1 = 1 element per block)
- `paral`: (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation
- `threads`: (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available
- `mixblock_size`: (optional) only for debug purpose
- `outgroup`: (optional) group name to store results from Crossprod inside hdf5 data file

**Value**

If all process is ok, returns a list with:

- "filename" File name where results are stored
- "dataset" route to results inside hdf5 data file
bdCrossprod_Weighted  

Matrix Crossprod with R-objects and Delayed Array Object

Description

This function performs a Crossproduct with weights matrix t(A)

Usage

bdCrossprod_Weighted(A, W, block_size = NULL, paral = NULL, threads = NULL)
**Arguments**

- **A**: a double matrix.
- **W**: a weighted matrix.
- **block_size**: (optional, default = 128) block size to make matrix multiplication, if `block_size = 1` no block size is applied (size 1 = 1 element per block).
- **paral**, (optional, default = TRUE) if `paral = TRUE` performs parallel computation else performs serial computation.
- **threads** (optional) only if `paral = true`, number of concurrent threads in parallelization.
  
  if threads is null then threads = maximum number of threads available.

**Value**

Matrix with t(A)

**Examples**

```r
# with numeric matrix
m <- 500
k <- 1500
n <- 400
A <- matrix(rnorm(n*k), nrow=n, ncol=k)
B <- matrix(rnorm(n*k), nrow=k, ncol=n)

# Serial execution
Serie <- bdCrossprod_Weighted(A, B, paral = FALSE)

# Parallel execution with 2 threads and blocks 256x256
Par_2cor <- bdCrossprod_Weighted(A, B, paral = TRUE, block_size = 256, threads = 2)
```

---

**bddtrsm**

*Solves matrix equations: A*X = B*

**Description**

Solves matrix equations: A*X = B

**Usage**

`bddtrsm(R, Z, threads = NULL)`

**Arguments**

- **R**: numerical matrix.
- **Z**: numerical matrix.
- **threads**: integer with number of threads to use with parallelized execution.
bdgetDatasetsList_hdf5

Value
X numerical matrix.

Examples
a <- "Unused function"

bdgetDatasetsList_hdf5

*Gets all dataset names inside a group*

Description
Gets a list of all dataset names inside a group or all the datasets names starting with a prefix under a group

Usage
bdgetDatasetsList_hdf5(filename, group, prefix = NULL)

Arguments
- filename, character array indicating the name of the file to create
- group, character array indicating the input group where the data sets are stored
- prefix, character array optional, indicates the prefix with which the dataset names begin, if null, then the function returns all datasets inside the group

Value
Full matrix with results from reduction

bdgetDim_hdf5

*Get dataset dimensions*

Description
get dataset dimensions

Usage
bdgetDim_hdf5(filename, element)
bdget_maf_hdf5

Arguments

filename, character array indicating the name of the file to create
element path to element, character array indicating the complete route to the element to query size (folder or dataset).

Value

none

Examples

matA <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15), nrow = 3, byrow = TRUE)
bCreate_hdf5_matrix_file("BasicMatVect.hdf5", matA, "INPUT", "matA")
bgetDim_hdf5("BasicMatVect.hdf5", "INPUT/matA")

# Remove file (used as example)
if (file.exists("BasicMatVect.hdf5")) {
  # Delete file if it exist
  file.remove("BasicMatVect.hdf5")
}

bdget_maf_hdf5

Get minor allele frequency

Description

This function normalize data scaling, centering or scaling and centering in a dataset stored in hdf5 file

Usage

bdget_maf_hdf5(
  filename,
  group,
  dataset,
  byrows = NULL,
  bparallel = NULL,
  wsize = NULL
)
bdImportData_hdf5

Arguments

- **filename**: string file name where dataset to normalize is stored
- **group**: string Matrix
- **dataset**: string Matrix
- **byrows**: boolean, default TRUE. If true, the frequency is calculated by rows, else, if byrows= FALSE, frequency is calculated by columns
- **bparallel**: boolean, Perform calculous in parallel?, by default TRUE.
- **wsize**: integer (default = 1000), file block size to read to perform normalization

Value

Numeric vector with allele frequencies

Examples

```r
library(BigDataStatMeth)

maf_cols = resc <- bdget_maf_hdf5("/Users/mailos/tmp/test/test.hdf5",
   "test", "mat1", byrows = FALSE )
maf_rows = resc <- bdget_maf_hdf5("/Users/mailos/tmp/test/test.hdf5",
   "test", "mat1", byrows = TRUE )
```

bdImportData_hdf5 Import data from url or a file

Description

This function download data from an url and decompress data (if needed), then imports the file to hdf5 data file

Usage

```r
bdImportData_hdf5(
   inFile,
   destFile,
   destGroup,
   destDataset,
   header = TRUE,
   rownames = FALSE,
   overwrite = FALSE,
   sep = NULL
)
```
bdImport_text_to_hdf5

Arguments

inFile: string file name or url with data to import
destFile: file name and path to store imported data
destGroup: group name to store the dataset
destDataset: dataset name to store the input file in hdf5
header: (optional) either a logical value indicating whether the column names of x are to be written along with x, or a character vector of column names to be written. See the section on ‘CSV files’ for the meaning of col.names = NA.
rownames: (optional) either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written.
overwrite: (optional) either a logical value indicating whether the output file can be overwritten or not.
sep: (optional), by default = "\t". The field separator string. Values within each row of x are separated by this string.

Value
	none value returned, data are stored in a dataset inside an hdf5 data file.

Examples

print("Example in vignette")

bdImport_text_to_hdf5: Converts text file to hdf5 data file

Description

Converts text file to hdf5 data file

Usage

bdImport_text_to_hdf5(
  filename,
  outputfile,
  outGroup,
  outDataset,
  sep = NULL,
  header = FALSE,
  rownames = FALSE,
  overwrite = FALSE
)
bdImpute_snps_hdf5

Impute SNPs in hdf5 omic dataset

Usage

bdImpute_snps_hdf5(
  filename,
  group,
  dataset,
  outgroup = NULL,
  outdataset = NULL,
  bycols = TRUE
)

Arguments

filename, character array indicating the name of the file to create
group, character array indicating the input group where the data set to be imputed is.
dataset, character array indicating the input dataset to be imputed
bdInvCholesky

outgroup, optional character array indicating group where the data set will be saved after imputation if ‘outgroup’ is NULL, output dataset is stored in the same input group.

outdataset, optional character array indicating dataset to store the resulting data after imputation if ‘outdataset’ is NULL, input dataset will be overwritten.

bycls, boolean by default = true, true indicates that the imputation will be done by columns, otherwise, the imputation will be done by rows.

Value

Original hdf5 data file with imputed data

bdInvCholesky | Inverse Cholesky

Description

This function get the inverse of a numerical matrix. If x is hermitian and positive-definite matrix then gets the inverse using Cholesky decomposition.

Usage

bdInvCholesky(X)

Arguments

X numerical matrix. If x is Hermitian and positive-definite performs

Value

inverse matrix of d

Examples

A <- matrix(c(3,4,3,4,8,6,3,6,9), byrow = TRUE, ncol = 3)
bdInvCholesky(A)
**bdlm_paral**  
*Linear regression using MLR-MR algorithm*

Description

Linear regression for Big Data using MLR-MR algorithm to perform lm regression with big data:  

Usage

```
bdlm_paral(Y, model, blocks, threads = NULL)
```

Arguments

- **Y**, numerical matrix column with response variable
- **model**, numerical matrix with paired observations of the predictor variable X
- **blocks**, integer with number of blocks we want to split matrix if null matrix is split in blocks as maximum of 1000 variables per block
- **threads**, threads (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available

Value

Lineal regression coefficients

Examples

```r  
## Not run:  
library(BigDataStatMeth)  
data(mtcars)  
  Y <- mtcars$mpg  
  X <- model.matrix(~ wt + cyl, data=mtcars)  
  m <- 4  
  res <- bdlm_paral(X, Y, m, 1)  
  res  

## End(Not run)
```
bdNormalize_Data  Normalize Delayed Array matrix

Description

This function performs a numerical or Delayed Array matrix normalization

Usage

bdNormalize_Data(X, bcenter = NULL, bscale = NULL)

Arguments

X  numerical or Delayed Array Matrix
bcenter  logical (default = TRUE) if TRUE, centering is done by subtracting the column means
bscale  logical (default = TRUE) if TRUE, centering is done by subtracting the column means

Value

numerical matrix

Examples

m <- 500
n <- 100
x <- matrix(rnorm(m*n), nrow=m, ncol=n)
# with numeric matrix
bdNormalize_Data(x)
# Only scale
bdNormalize_Data(x, bcenter = FALSE)
# Only center
bdNormalize_Data(x, bscale = FALSE)
bdNormalize_hdf5
Normalize dataset in hdf5 file

**Description**

This function normalizes data scaling, centering or scaling and centering in a dataset stored in hdf5 file.

**Usage**

```r
bdNormalize_hdf5(
  filename, 
  group, 
  dataset, 
  bcenter = NULL, 
  bscale = NULL, 
  wsize = NULL, 
  force = FALSE 
)
```

**Arguments**

- `filename`: string file name where dataset to normalize is stored
- `group`: string Matrix
- `dataset`: string Matrix
- `bcenter`: logical (default = TRUE) if TRUE, centering is done by subtracting the column means
- `bscale`: logical (default = TRUE) if TRUE, centering is done by subtracting the column means
- `wsize`: integer (default = 1000), file block size to read to perform normalization
- `force`: boolean if true, previous results in same location inside hdf5 will be overwritten.

**Value**

file with scaled, centered or scaled and centered dataset

**Examples**

```r
a = "See vignette"
```
**bdparallelpow2**  
*Pow vector*

**Description**  
Gets pow2 vector

**Usage**  
bdparallelpow2(x)

**Arguments**  
x  numerical vector

**Value**  
Numeric Vector

**Examples**

```r
library(BigDataStatMeth)

n <- 100
x <- rnorm(n)

# with numeric matrix
res <- bdparallelpow2(x)
```

---

**bdparallelVectorSum**  
*Sumarize vector*

**Description**  
This function summarize the elements of a vector

**Usage**  
bdparallelVectorSum(x)

**Arguments**  
x  numerical vector

**Value**  
none value returned, data are stored in a dataset inside an hdf5 data file.
Examples

library(BigDataStatMeth)

n <- 100
x <- rnorm(n)

# with numeric matrix
res <- bdparallelVectorSum(x)

bdPCA_hdf5

PCA Descomposition

Description

Compute PCA

Usage

bdPCA_hdf5(
  filename,     # string, file name where dataset is stored
  group,       # string group name where dataset is stored in file
  dataset,     # string dataset name with data to perform PCA
  ncomponents = 0L,  # integer, number of components to be computed, by default ncomponents = 0, all components are computed
  bcenter = FALSE,  # logical value if true data is centered to zero
  bscale = FALSE,  # logical value, if true data is scaled
  k = 2L,  # number of local SVDs to concatenate at each level, performance parameter
  q = 1L,  # number of levels to compute SVD for PCA, performance parameter
  rankthreshold = 0,  # rank threshold for SVD computation
  force = FALSE,  # force computation
  threads = NULL  # number of threads to use
)

Arguments

filename  string, file name where dataset is stored

group  string group name where dataset is stored in file

dataset  string dataset name with data to perform PCA

ncomponents  integer, number of components to be computed, by default ncomponents = 0, all components are computed

bcenter  logical value if true data is centered to zero

bscale  logical value, if true data is scaled

k  number of local SVDs to concatenate at each level, performance parameter

q  number of levels to compute SVD for PCA, performance parameter
rankthreshold  double, threshold used to determine the range of the array. The matrix rank is equal to the number of singular values different from the threshold. By default, threshold = 0 is used to get the matrix rank, but it can be changed to an approximation of 0.
force  logical value, if true, the SVD is forced to be computed although the SVD exists
threads  integer number of threads used to run PCA

Value
original file with results in folder PCA/<datasetname>

bdpseudoinv  Pseudo-Inverse

Description
Compute the pseudo-inverse of a singular matrix

Usage
bdpseudoinv(X)

Arguments
X  Singular matrix (m x n)

Value
Pseudo-inverse matrix of A

bdQR  QR Decomposition

Description
This function compute QR decomposition (also called a QR factorization) of a matrix A into a product A = QR of an orthogonal matrix Q and an upper triangular matrix R.

Usage
bdQR(X, thin = NULL)

Arguments
X  a real square matrix
thin  boolean thin, if thin = true returns Q thin decomposition else returns Q full decomposition, default thin = false
bdReduce_matrix_hdf5

Description

Reduce hdf5 datasets inside a group by rows or columns and store complete matrix inside hdf5 data file.

Usage

bdReduce_matrix_hdf5(
    filename,
    group,
    reducefunction,
    outgroup = NULL,
    outdataset = NULL,
    force = FALSE,
    remove = FALSE
)

Arguments

filename, character array indicating the name of the file to create
group, character array indicating the input group where the data sets are stored
reducefunction, single character with function to apply, can be '+' or '-'
outgroup, optional character array indicating group where the data set will be saved after
outdataset, optional character array indicating dataset to store the resulting data after
force, boolean if true, previous results in same location inside hdf5 will be overwritten.
remove, boolean if true, removes original matrices, by default bremove = false.

Value

Full matrix with results from reduction
## bdRemove_lowdata

### Description

Remove SNPs in hdf5 omic dataset with low data.

### Usage

```r
bdRemove_lowdata(filename, group, dataset, outgroup, outdataset, pcent, bycols)
```

### Arguments

- `filename`: character array indicating the name of the file to create.
- `group`: character array indicating the input group where the data set to be imputed is.
- `dataset`: character array indicating the input dataset to be imputed.
- `outgroup`: character array indicating group where the data set will be saved after remove data with if ‘outgroup’ is NULL, output dataset is stored in the same input group.
- `outdataset`: character array indicating dataset to store the resulting data after imputation if ‘outdataset’ is NULL, input dataset will be overwritten.
- `pcent`: by default pcent = 0.5. Numeric indicating the percentage to be considered to remove SNPs, SNPS with percentage equal or higest will be removed from data.
- `bycols`: boolean by default = true, if true, indicates that SNPs are in cols, if SNPincols = false indicates that SNPs are in rows.

### Value

Original hdf5 data file with imputed data.

## bdRemove_hdf5_element

### Description

Remove element group or dataset from hdf5 file.

### Usage

```r
bdRemove_hdf5_element(filename, element)
```

### Arguments

- `filename`: character array indicating the name of the file to create.
- `element`: path to element, character array indicating the complete route to the element to be removed (folder or dataset).
Value

none

Examples

```
matA <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15), nrow = 3, byrow = TRUE)
matB <- matrix(c(15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,5,3,4,5,2,6,2,3,4,
         42, 23, 23, 423,1,2), nrow = 3, byrow = TRUE)

bdCreate_hdf5_matrix_file("BasicMatVect.hdf5", matA, "INPUT", "matA")
bdAdd_hdf5_matrix(matB, "BasicMatVect.hdf5", "INPUT", "matB")

bdRemove_hdf5_element("BasicMatVect.hdf5", "INPUT/matA")
```

# Remove file (used as example)
if (file.exists("BasicMatVect.hdf5")) {
  # Delete file if it exist
  file.remove("BasicMatVect.hdf5")
}

---

**bdremove_maf_hdf5**  
Remove SNPs in hdf5 omic dataset with low data

**Description**

Remove SNPs in hdf5 omic dataset with low data

**Usage**

```
bdremove_maf_hdf5(
  filename,
  group,
  dataset,
  outgroup,
  outdataset,
  maf,
  bycols,
  blocksize
)
```

**Arguments**

- **filename**: character array indicating the name of the file to create
- **group**: character array indicating the input group where the data set to be imputed is.
- **dataset**: character array indicating the input dataset to be imputed
outgroup, character array indicating group where the data set will be saved after remove data with if ‘outgroup’ is NULL, output dataset is stored in the same input group.

outdataset, character array indicating dataset to store the resulting data after imputation if ‘outdataset’ is NULL, input dataset will be overwritten.

maf, by default maf = 0.05. Numeric indicating the percentage to be considered to remove SNPs, SNPS with highest MAF will be removed from data

bycols, boolean by default = true, if true, indicates that SNPs are in cols, if SNPincols = false indicates that SNPs are in rows.

blocksize, integer, block size dataset to read/write and calculate MAF, by default this operations is made in with 100 rows if byrows = true or 100 cols if byrows = false.

Value

Original hdf5 data file with imputed data

bdScalarwproduct  Matrix - Weighted Scalar Multiplication with numerical or DelayedArray data

Description

This function performs a weighted product of a matrix(X) with a weighted diagonal matrix (w)

Usage

bdScalarwproduct(A, w, op)

Arguments

A numerical or Delayed Array matrix
w scalar, weight
op string indicating if operation "Xw" or "wX"

Value

numerical matrix

Examples

```
n <- 100
p <- 60

X <- matrix(rnorm(n*p), nrow=n, ncol=p)
w <- 0.75

bdScalarwproduct(X, w,"Xw")
bScalarwproduct(X, w,"wX")
```
bdSolve  
Solve matrix equations

Description

This function solve matrix equations \( A \times X = B \) where \( A \) is an \( N \)-by-\( N \) matrix and \( X \) and \( B \) are \( N \)-by-\( NRHS \) matrices.

Usage

bdSolve(A, B)

Arguments

- **A**: numerical matrix.
- **B**: numerical matrix.

Value

- **X**: numerical matrix.

Examples

```
library(BigDataStatMeth)

n <- 500
m <- 500

# R Object
A <- matrix(runif(n*m), nrow = n, ncol = m)
B <- matrix(runif(n), nrow = n)
A5 <- A%*%t(A)

X <- bdSolve(A, B)
XR <- solve(A,B)
all.equal(X, XR, check.attributes=FALSE)
```
bdSplit_matrix_hdf5  

Description

Split hdf5 dataset by rows or columns and store splitted submatrices inside hdf5 file.

Usage

bdSplit_matrix_hdf5(
  filename,
  group, 
  dataset, 
  outgroup = NULL, 
  outdataset = NULL, 
  nblocks = NULL, 
  blocksize = NULL, 
  bycols = TRUE, 
  force = FALSE
)

Arguments

filename, character array indicating the name of the file to create

group, character array indicating the input group where the data set to be imputed is.

dataset, character array indicating the input dataset to be imputed

outgroup, optional character array indicating group where the data set will be saved after imputation if ‘outgroup’ is NULL, output dataset is stored in the same input group.

outdataset, optional character array indicating dataset to store the resulting data after imputation if ‘outdataset’ is NULL, input dataset will be overwritten.

nblocks, integer number of blocks in which we want to split the data

blocksize, integer, number of elements in each block

bycols, boolean by default = true, true indicates that the imputation will be done by columns, otherwise, the imputation will be done by rows

force, boolean if true, previous results in same location inside hdf5 will be overwritten.

Value

Original hdf5 data file with imputed data
bdSVD

\textit{k first SVD components for DelayedArray}

\textbf{Description}

This function gets \( k \) first components from svd decomposition of numerical or Delayed Array.

\textbf{Usage}

\[ \text{bdSVD}(X, k = 0L, \text{nev} = 0L, \text{bcenter} = \text{TRUE}, \text{bscale} = \text{TRUE}) \]

\textbf{Arguments}

\begin{itemize}
  \item \textbf{X} numerical or Delayed Array matrix
  \item \textbf{k} number of eigen values, this should satisfy \( k = \min(n, m) - 1 \)
  \item \textbf{nev} (optional, default \text{nev} = \( n-1 \)) Number of eigenvalues requested. This should satisfy \( 1 \leq \text{nev} \leq n \), where \( n \) is the size of matrix.
  \item \textbf{bcenter} (optional, default = \text{TRUE}). If center is \text{TRUE} then centering is done by subtracting the column means (omitting NAs) of \( x \) from their corresponding columns, and if center is \text{FALSE}, no centering is done.
  \item \textbf{bscale} (optional, default = \text{TRUE}). If scale is \text{TRUE} then scaling is done by dividing the (centered) columns of \( x \) by their standard deviations if center is \text{TRUE}, and the root mean square otherwise. If scale is \text{FALSE}, no scaling is done.
\end{itemize}

\textbf{Value}

\begin{itemize}
  \item \( u \) eigenvectors of \( AA^{t} \), \( mxn \) and column orthogonal matrix
  \item \( v \) eigenvectors of \( A^{t}A \), \( nxn \) orthogonal matrix
  \item \( d \) singular values, \( nxn \) diagonal matrix (non-negative real values)
\end{itemize}

\textbf{Examples}

\begin{verbatim}
  n <- 500
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)

  # svd without normalization
  decsvd <- bdSVD( A, bscale = FALSE, bcenter = FALSE ) # No matrix normalization
  decsvd$d
decsvd$u

  # svd with normalization
  decsvd <- bdSVD( A, bscale = TRUE, bcenter = TRUE ) # Matrix normalization
  decsvd$d
decsvd$u
\end{verbatim}
# svd with scaled matrix (sd)
decsvd <- bdSVD( A, bscale = TRUE, bcenter = FALSE) # Scaled matrix

decsvd$d
decsvd$u

# svd with centered matrix (sd)
decsvd <- bdSVD( A, bscale = FALSE, bcenter = TRUE) # Centered matrix
decsvd$d
decsvd$u

bdSVD_hdf5

Block SVD decomposition using an incremental algorithm.

Description
Block SVD decomposition for hdf5 files using an incremental algorithm.

Usage
bdSVD_hdf5(
  file,
  group = NULL,
  dataset = NULL,
  k = 2L,
  q = 1L,
  bcenter = TRUE,
  bscale = TRUE,
  rankthreshold = 0,
  threads = NULL
)

Arguments
file a real nxp matrix in hdf5 file

group group in hdf5 data file where dataset is located
dataset matrix dataset with data to perform SVD
k number of local SVDs to concatenate at each level
q number of levels
bcenter (optional, default = TRUE) . If center is TRUE then centering is done by subtracting the column means (omitting NAs) of x from their corresponding columns, and if center is FALSE, no centering is done.
bscale (optional, default = TRUE) . If scale is TRUE then scaling is done by dividing the (centered) columns of x by their standard deviations if center is TRUE, and the root mean square otherwise. If scale is FALSE, no scaling is done.
**rankthreshold**

- **Details**
  
  Singular values and left singular vectors of a real nxp matrix

**Value**

- a list of three components with the singular values and left and right singular vectors of the matrix

A List with:

- "u" eigenvectors of $AA^t$, mxn and column orthogonal matrix
- "v" eigenvectors of $A^tA$, nxn orthogonal matrix
- "v" singular values, nxn diagonal matrix (non-negative real values)

**Usage**

- bdSVD_lapack(X, bcenter = TRUE, bscale = TRUE, complete = FALSE)

**Arguments**

- **X**
  - numerical or Delayed Array matrix

- **bcenter**
  - (optional, default = TRUE). If center is TRUE then centering is done by subtracting the column means (omitting NAs) of x from their corresponding columns, and if center is FALSE, no centering is done.

- **bscale**
  - (optional, default = TRUE). If scale is TRUE then scaling is done by dividing the (centered) columns of x by their standard deviations if center is TRUE, and the root mean square otherwise. If scale is FALSE, no scaling is done.

- **complete**
  - (optional, default = FALSE). If complete is TRUE svd function returns complete u and v
**Value**

- u eigenvectors of $AA^t$, mxn and column orthogonal matrix
- v eigenvectors of $A^tA$, nxn orthogonal matrix
- d singular values, nxn diagonal matrix (non-negative real values)

**Examples**

```r
library(BigDataStatMeth)
n <- 500
A <- matrix(rnorm(n*n), nrow=n, ncol=n)

# svd without normalization
decsvd <- bdSVD_lapack( A, bscale = FALSE, bcenter = FALSE ) # No matrix normalization
decsvd$d
decsvd$u

# svd with normalization
decsvd <- bdSVD_lapack( A, bscale = TRUE, bcenter = TRUE) # Matrix normalization
decsvd$d
decsvd$u

decsvd <- bdSVD_lapack( A ) # Matrix normalization too

# svd with scaled matrix (sd)
decsvd <- bdSVD_lapack( A, bscale = TRUE, bcenter = FALSE) # Scaled matrix
```

**bdtCrossprod**

**Crossproduct**

**Description**

This function performs a transposed crossproduct of numerical matrix.

**Usage**

```
bdtCrossprod(A, B = NULL, block_size = 256, paral = TRUE, threads = NULL)
```

**Arguments**

- **A** numerical or Delayed Array matrix
- **B** optional, numerical or Delayed Array matrix
- **block_size** (optional, default = 128) block size to make matrix multiplication, if `block_size = 1` no block size is applied (size 1 = 1 element per block)
- **paral,** (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation
- **threads** (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available
Value

numerical matrix with crossproduct

Examples

```r
n <- 100
p <- 60
X <- matrix(rnorm(n*p), nrow=n, ncol=p)

# without DelayedArray
bdtCrossprod(X)

all.equal(crossprod(X), bdtCrossprod(X))

n <- 100
p <- 100
Y <- matrix(rnorm(n*p), nrow=n)

bdtCrossprod(X,Y)
```

---

**bdtCrossprod_hdf5**  
*Transposed Crossprod with hdf5 matrix*

### Description

This function performs the transposed crossprod from a matrix inside and hdf5 data file

### Usage

```r
bdtCrossprod_hdf5(
    filename,  
    group,  
    A,  
    groupB = NULL,  
    B = NULL,  
    block_size = NULL,  
    paral = NULL,  
    threads = NULL,  
    mixblock_size = NULL,  
    outgroup = NULL
)
```
bdtCrossprod_hdf5

Arguments

- **filename**: string file name where dataset to normalize is stored
- **group A**: string, group name where dataset A is stored
- **group B**: string, group name where dataset B is stored
- **block_size**: (optional, default = 128) block size to make matrix multiplication, if 'block_size = 1' no block size is applied (size 1 = 1 element per block)
- **paral**: (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation
- **threads**: (optional) only if paral = true, number of concurrent threads in parallelization
- **mixblock_size**: (optional) only for debug purpose
- **outgroup**: (optional) group name to store results from Crossprod inside hdf5 data file

Value

If all process is ok, returns a list with:

- "filename" File name where results are stored
- "dataset" route to results inside hdf5 data file

Examples

```r
library(BigDataStatMeth)
library(rhdf5)

matA <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15), nrow = 3, byrow = TRUE)
matB <- matrix(c(15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,5,3,4,5,2,6,2,3,4,
                 42, 23, 23, 423,1,2), ncol = 5, byrow = TRUE)

bdCreate_hdf5_matrix_file("BasicMatVect.hdf5", matA, "INPUT", "matA")
bAdd_hdf5_matrix( matB, "BasicMatVect.hdf5", "INPUT", "matB")

res <- bdtCrossprod_hdf5("BasicMatVect.hdf5", "INPUT",
                         "matA", block_size = 2)
res2 <- bdtCrossprod_hdf5("BasicMatVect.hdf5", "INPUT",
                          "matA", "INPUT","matB", block_size = 2)

# Open file
h5fdelay = H5Fopen("BasicMatVect.hdf5")

res <- h5fdelay$OUTPUT$tCrossProd_matAxmatA
res2 <- h5fdelay$OUTPUT$tCrossProd_matAxmatB

all.equal(tcrossprod(matA), res)
all.equal(tcrossprod(matA, matB), res2)
```
bdtcrossprod_weighted

Block matrix multiplication with Delayed Array Object

Description
This function performs a Crossproduct with weights matrix A

Usage
bdtcrossprod_weighted(A, W, block_size = NULL, paral = NULL, threads = NULL)

Arguments
- **A**: a double matrix.
- **W**: a Weighted matrix
- **block_size**: (optional, default = 128) block size to make matrix multiplication, if `block_size = 1` no block size is applied (size 1 = 1 element per block)
- **paral**, (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation
- **threads**: (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available

Value
Matrix with A

Examples

```r
# with numeric matrix
m <- 500
k <- 1500
n <- 400
A <- matrix(rnorm(n*k), nrow=n, ncol=k)
B <- matrix(rnorm(n*k), nrow=k, ncol=n)

# Serial execution
```
bdwproduct

Matrix - Weighted vector Multiplication with numerical or Delayed Array data

Description

This function performs a weighted product of a matrix(X) with a weighted diagonal matrix (w)

Usage

bdwproduct(X, w, op)

Arguments

X numerical or Delayed Array matrix
w vector with weights
op string indicating if operation 'XtwX' and 'XwXt' for weighted cross product (Matrix - Vector - Matrix) or 'Xw' and 'wX' for weighted product (Matrix - Vector)

Value

numerical matrix

Examples

n <- 100
p <- 60
X <- matrix(rnorm(n*p), nrow=n, ncol=p)
u <- runif(n)
w <- u * (1 - u)
ans <- bdwproduct(X, w,"xtwx")

Serie <- bdtCrossprod_Weighted(A, B, paral = FALSE)

# Parallel execution with 2 threads and blocks 256x256
Par_2cor <- bdtCrossprod_Weighted(A, B, paral = TRUE, block_size = 256, threads = 2)
BigDataStatMeth

BigDataStatMEth package documentation

Description

BigDataStatMEth package documentation

blockmult

Block matrix multiplication with Delayed Array Object

Description

This function performs a block matrix-matrix multiplication with numeric matrix or Delayed Arrays

Usage

blockmult(
  a,
  b,
  block_size = NULL,
  paral = NULL,
  threads = NULL,
  bigmatrix = NULL,
  mixblock_size = NULL,
  outfile = NULL,
  onmemory = NULL
)

Arguments

a
  a double matrix.

b
  a double matrix.

block_size
  (optional, default = 128) block size to make matrix multiplication, if ‘block_size = 1’ no block size is applied (size 1 = 1 element per block)

paral,
  (optional, default = TRUE) if paral = TRUE performs parallel computation else performs seria computation

threads
  (optional) only if bparal = true, number of concurrent threads in parallelization if threads is null then threads = maximum number of threads available

bigmatrix
  (optiona, default = 5000) maximum number of rows or columns to consider as big matrix and work with hdf5 files, by default a matrix with more than 5000 rows or files is considered big matrix and computation is made in disk
**mixblock_size** (optional, default = 128), only if we are working with big matrix and parallel computation = true. Block size for mixed computation in big matrix parallel. Size of the block to be used to perform parallelized memory memory of the block read from the disk being processed.

**outfile** (optional) file name to work with hdf5 if we are working with big matrix in disk.

**onmemory** (optional) if onmemory = TRUE the multiplication is forced to execute in memory.

### Value

A List with:

- "matrix" Result matrix if execution has been performed in memory
- "filename" HDF5 filename if execution has been performed in disk, HDF5 file contains:
  - "INPUT"hdf5 group with input matrix A and B
  - "OUTPUT"hdf5 group with output matrix C

with input and output matrix.

### Examples

```r
# with numeric matrix
m <- 500
k <- 1500
n <- 400
A <- matrix(rnorm(n*k), nrow=n, ncol=k)
B <- matrix(rnorm(n*k), nrow=k, ncol=n)
C <- blockmult(A,B,128, TRUE)
```

---

### cancer

**Cancer classification**

**Description**

A three factor level variable corresponding to cancer type

**Usage**

data(cancer)

**Format**

factor level with three levels
cancer factor with cancer type
Examples

```r
data(cancer)
```

dataset colesterol

**Description**

This is a dataset containing simulated variables related to colesterol.

colesterol.csv

- **TCholesterol** TCholesterol
- **Age** Age
- **Insulin** Insulin
- **Creatinine** Creatinine
- **BUN** BUN
- **LLDR** LLDR
- **Triglycerides** Triglycerides
- **HDL_C** HDL_C
- **LDL_C** LDL_C
- **Sex** Sex

This data is used in bdImport_text_to_hdf5().

getQRbyBlocks

**Description**

This function is an application of the BigDataStatMeth functions to generate new methods. This function perform a QR from two matrices stored in hdf5 data file. This function applies matrix partitioning, merge bloks to create a full matrix, apply a function to different blocks...

**Usage**

```r
getQRbyBlocks(strdataset, file, mblocks, center, scale, bcols, overwrt)
```
Arguments

strdataset: string, dataset path within the hdf5 data file from which we want to calculate the QR.

file: string file name where dataset to normalize is stored.

mblocks: number of blocks in which we want to partition the matrix to perform the calculations.

center, scale: boolean, if true, dataset is centered to perform calculus.

bcols: boolean if bcols = TRUE matrix it’s split by columns if bcols = FALSE, then matrix or dataset is split by rows.

overwrt: boolean, if true, datasets existing inside a file must be overwritten if we are using the same names.

Value

hdf5 data file with CCA results.

Examples

print("Example in vignette")

---

miRNA | miRNA

Description

A three factor level variable corresponding to cancer type.

Usage

data(miRNA)

Format

Dataframe with 21 samples and 537 variables.

columns: variables

rows: samples

Examples

data(miRNA)
Index

* datasets
  cancer, 48
  miRNA, 50

bdAdd_hdf5_matrix, 3
bdapplyFunction_hdf5, 4
bdBind_hdf5, 5
bdblockmult, 6
bdblockmult_hdf5, 7
bdblockmult_sparse, 9
bdblockmult_sparse_hdf5, 10
bdblockmult_vector, 11
bdCCA_hdf5, 12
bdCreate_hdf5_matrix, 16
bdCreateEmptyDataset_hdf5, 13
bdCreateGroup_hdf5, 14
bdCreateLink_hdf5, 15
bdCrossprod, 17
bdCrossprod_hdf5, 18
bdCrossprodWeighted, 19
bddtrsm, 20
bdget_maf_hdf5, 22
bdgetDatasetsList_hdf5, 21
bdgetDim_hdf5, 21
bdImport_text_to_hdf5, 24
bdImportData_hdf5, 23
bdImpute_snps_hdf5, 25
bdInvCholesky, 26
bdlm_paral, 27
bdNormalze_Data, 28
bdNormalize_hdf5, 29
bdparallelpow2, 30
bdparallelVectorSum, 30
bdPCA_hdf5, 31
bdpsudoinv, 32
bdQR, 32
bdReduce_matrix_hdf5, 33
bdRemove_hdf5_element, 34
bdremove_maf_hdf5, 35
bdRemovelowdata, 34

bdScalarwproduct, 36
bdSolve, 37
bdSplit_matrix_hdf5, 38
bdSVD, 39
bdSVD_hdf5, 40
bdSVD_lapack, 41
bdtCrossprod, 42
bdtCrossprod_hdf5, 43
bdtCrossprodWeighted, 45
bdwproduct, 46
BigDataStatMeth, 47
blockmult, 47

cancer, 48
colesterol, 49

getQRbyBlocks, 49

miRNA, 50