Package ‘BigDataStatMeth’

November 29, 2025

Type Package

Title Tools and Infrastructure for Developing 'Scalable' 'HDF5'-Based Methods

Version 1.0.2

Date 2025-11-28

Description A framework for 'scalable' statistical computing on large on-disk matrices stored in 'HDF5' files. It provides efficient block-wise implementations of core linear-algebra operations (matrix multiplication, SVD, PCA, QR decomposition, and canonical correlation analysis) written in C++ and R. These building blocks are designed not only for direct use, but also as foundational components for developing new statistical methods that must operate on datasets too large to fit in memory. The package supports data provided either as 'HDF5' files or standard R objects, and is intended for high-dimensional applications such as 'omics' and precision-medicine research.

License MIT + file LICENSE

Depends R (>= 4.1.0)

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

LinkingTo Rcpp, RcppEigen, Rhdf5lib, BH

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

SystemRequirements GNU make, C++17

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.3

NeedsCompilation yes

Author Dolors Pelegri-Siso [aut, cre] (ORCID: <https://orcid.org/0000-0002-5993-3003>), Juan R. Gonzalez [aut] (ORCID: <https://orcid.org/0000-0003-3267-2146>)

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

Repository CRAN

Date/Publication 2025-11-29 13:30:28 UTC

Contents

`bdapply_Function_hdf5` *Apply function to different datasets inside a group*

Description

This function provides a unified interface for applying various mathematical operations to HDF5 datasets. It supports both single-dataset operations and operations between multiple datasets.

Usage

```
bdapply_Function_hdf5(
  filename,
  group,
  datasets,
  outgroup,
  func,
  b_group = NULL,
  b_datasets = NULL,
  overwrite = FALSE,
  transp_dataset = FALSE,
  transp_bdataset = FALSE,
  fullMatrix = FALSE,
  byrows = FALSE,
  threads = 2L
)
```

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 NULL, output dataset is stored in the same input group
`func`	Character array, function to be applied: - "QR": QR decomposition via bdQR() - "CrossProd": Cross product via bdCrossprod() - "tCrossProd": Transposed cross product via bdtCrossprod() - "invChol": Inverse via Cholesky decomposition - "blockmult": Matrix multiplication - "CrossProd_double": Cross product with two matrices - "tCrossProd_double": Transposed cross product with two matrices - "solve": Matrix equation solving - "sdmean": Standard deviation and mean computation
`b_group`	Optional character array indicating the input group for secondary datasets (used in two-matrix operations)
`b_datasets`	Optional character array indicating the secondary datasets for two-matrix operations
`overwrite`	Optional boolean. If true, overwrites existing results
`transp_dataset`	Optional boolean. If true, transposes first dataset
`transp_bdataset`	Optional boolean. If true, transposes second dataset
`fullMatrix`	Optional boolean for Cholesky operations. If true, stores complete matrix; if false, stores only lower triangular
`byrows`	Optional boolean for statistical operations. If true, computes by rows; if false, by columns
`threads`	Optional integer specifying number of threads for parallel processing

Details

//' For matrix multiplication operations (`blockmult`, `CrossProd_double`, `tCrossProd_double`), the `datasets` and `b_datasets` vectors must have the same length. Each operation is performed element-wise between the corresponding pairs of datasets. Specifically, the `b_datasets` vector defines the second operand for each matrix multiplication. For example, if `datasets = {"A1", "A2", "A3"}` and `b_datasets = {"B1", "B2", "B3"}`, the operations executed are: `A1 %*% B1`, `A2 %*% B2`, and `A3 %*% B3`.

Example: If `datasets = {"A1", "A2", "A3"}` and `b_datasets = {"B1", "B2", "B3"}`, the function computes: `A1 %*% B1`, `A2 %*% B2`, and `A3 %*% B3`

Value

Modifies the HDF5 file in place, adding computed results

Note

Performance is optimized through: - Block-wise processing for large datasets - Parallel computation where applicable - Memory-efficient matrix operations

Examples

```
## Not run:
# Create a sample large matrix in HDF5
# Create hdf5 datasets
bdCreate_hdf5_matrix(filename = "test_temp.hdf5",
                    object = Y, group = "data", dataset = "Y",
                    transp = FALSE,
                    overwriteFile = TRUE, overwriteDataset = TRUE,
                    unlimited = FALSE)

bdCreate_hdf5_matrix(filename = "test_temp.hdf5",
                    object = X,  group = "data",  dataset = "X",
                    transp = FALSE,
                    overwriteFile = FALSE, overwriteDataset = TRUE,
                    unlimited = FALSE)

bdCreate_hdf5_matrix(filename = "test_temp.hdf5",
                    object = Z,  group = "data",  dataset = "Z",
                    transp = FALSE,
                    overwriteFile = FALSE, overwriteDataset = TRUE,
                    unlimited = FALSE)

dsets <- bdgetDatasetsList_hdf5("test_temp.hdf5", group = "data")
dsets

# Apply function :  QR Decomposition
bdapply_Function_hdf5(filename = "test_temp.hdf5",
                     group = "data",datasets = dsets,
                     outgroup = "QR",func = "QR",
                     overwrite = TRUE)

## End(Not run)
```

bdBind_hdf5_datasets *Bind matrices by rows or columns*

Description

This function merges existing matrices within an HDF5 data file either by combining their rows (stacking vertically) or columns (joining horizontally). It provides functionality similar to R's rbind and cbind operations.

Usage

```
bdBind_hdf5_datasets(
  filename,
  group,
  datasets,
  outgroup,
  outdataset,
  func,
  overwrite = FALSE
)
```

Arguments

`filename`	Character array indicating the name of the file to create
`group`	Character array indicating the input group containing the datasets
`datasets`	Character array specifying the input datasets to bind
`outgroup`	Character array indicating the output group for the merged dataset. If NULL, output is stored in the same input group
`outdataset`	Character array specifying the name for the new merged dataset
`func`	Character array specifying the binding operation: - "bindRows": Merge datasets by rows (vertical stacking) - "bindCols": Merge datasets by columns (horizontal joining) - "bindRowsbyIndex": Merge datasets by rows using an index
`overwrite`	Boolean indicating whether to overwrite existing datasets. Defaults to false

Details

The function performs dimension validation before binding:

- For row binding: All datasets must have the same number of columns
- For column binding: All datasets must have the same number of rows

Memory efficiency is achieved through:

- Block-wise reading and writing
- Minimal data copying
- Proper resource cleanup

Value

A list containing the location of the combined dataset:

fn Character string. Path to the HDF5 file containing the result

ds Character string. Full dataset path to the bound/combined dataset within the HDF5 file

Note

When binding by rows with an index, the index determines the order of combination

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
a <- matrix(1:12, 4, 3)
b <- matrix(13:24, 4, 3)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", a, "data", "A")
bdCreate_hdf5_matrix("test.hdf5", b, "data", "B")

# Bind by rows
bdBind_hdf5_datasets("test.hdf5", "data",
                     c("A", "B"),
                     "results", "combined",
                     "bindRows")

## End(Not run)
```

`bdblockMult` *Block-Based Matrix Multiplication*

Description

Performs efficient matrix multiplication using block-based algorithms. The function supports various input combinations (matrix-matrix, matrix-vector, vector-vector) and provides options for parallel processing and block-based computation.

Usage

```
bdblockMult(
  A,
  B,
  block_size = NULL,
  paral = NULL,
  byBlocks = TRUE,
  threads = NULL
)
```

Arguments

`A`	Matrix or vector. First input operand.
`B`	Matrix or vector. Second input operand.
`block_size`	Integer. Block size for computation. If NULL, uses maximum allowed block size.

`paral`	Logical. If TRUE, enables parallel computation. Default is FALSE.
`byBlocks`	Logical. If TRUE (default), forces block-based computation for large matrices. Can be set to FALSE to disable blocking.
`threads`	Integer. Number of threads for parallel computation. If NULL, uses half of available threads or maximum allowed threads.

Details

This function implements block-based matrix multiplication algorithms optimized for cache efficiency and memory usage. Key features:

- Input combinations supported:
  - Matrix-matrix multiplication
  - Matrix-vector multiplication (both left and right)
  - Vector-vector multiplication
- Performance optimizations:
  - Block-based computation for cache efficiency
  - Parallel processing for large matrices
  - Automatic block size selection
  - Memory-efficient implementation

The function automatically selects the appropriate multiplication method based on input types and sizes. For large matrices (>2.25e+08 elements), block-based computation is used by default.

Value

Matrix or vector containing the result of A * B.

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- Kumar, V. et al. (1994). Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publishing Company.

See Also

- `bdblockSum` for block-based matrix addition
- `bdblockSubstract` for block-based matrix subtraction

Examples

```
## Not run:
library(BigDataStatMeth)

# Matrix-matrix multiplication
N <- 2500
M <- 400
```

```
nc <- 4

set.seed(555)
mat <- matrix(rnorm(N*M, mean=0, sd=10), N, M)

# Parallel block multiplication
result <- bdblockMult(mat, mat,
                      paral = TRUE,
                      threads = nc)

# Matrix-vector multiplication
vec <- rnorm(M)
result_mv <- bdblockMult(mat, vec,
                         paral = TRUE,
                         threads = nc)

## End(Not run)
```

`bdblockmult_hdf5` *Hdf5 datasets multiplication*

Description

The bdblockmult_hdf5 function performs block-wise matrix multiplication between two matrices stored in an HDF5 file. This approach is also efficient for large matrices that cannot be fully loaded into memory.

Usage

```
bdblockmult_hdf5(
  filename,
  group,
  A,
  B,
  groupB = NULL,
  transpose_A = NULL,
  transpose_B = NULL,
  block_size = NULL,
  paral = NULL,
  threads = NULL,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	string specifying the path to the HDF5 file
`group`	string specifying the group within the HDF5 file containing matrix A.
`A`	string specifying the dataset name for matrix A. the data matrix to be used in calculus
`B`	string specifying the dataset name for matrix B.
`groupB`	string, (optional), An optional string specifying the group for matrix B. Defaults to the value of `group` if not provided.
`transpose_A`	Whether to transpose matrix A
`transpose_B`	Whether to transpose matrix B
`block_size`	integer (optional), an optional parameter specifying the block size for processing the matrices. If not provided, a default block size is used. The block size should be chosen based on the available memory and the size of the matrices
`paral`	boolean (optional), an optional parameter to enable parallel computation. Defaults to FALSE. Set `paral = true` to force parallel execution
`threads`	integer (optional), an optional parameter specifying the number of threads to use if paral = TRUE. Ignored if paral = FALSE.
`outgroup`	string (optional), An optional parameter specifying the group where the output matrix will be stored. If NULL, the output will be stored in the default group "OUTPUT".
`outdataset`	string (optional), An optional parameter specifying the dataset name for the output matrix. If NULL, the default name will be constructed as the name of dataset A concatenated with *x* and the name of dataset B.
`overwrite`	logical (optional), An optional parameter to indicate whether existing results in the HDF5 file should be overwritten. Defaults to FALSE. If FALSE and the dataset already exists, an error will be displayed, and no calculations will be performed. If TRUE and a dataset with the same name as specified in outdataset already exists, it will be overwritten.

Details

- The function `bdblockmult_hdf5()` is efficient for both matrices that cannot fit into memory (by processing in blocks) and matrices that can be fully loaded into memory, as it optimizes computations based on available resources.
- Ensure that the dimensions of `A` and `B` matrices are compatible for matrix multiplication.
- The `block size` should be chosen based on the available memory and the size of the matrices.
- If `bparal = true`, number of concurrent threads in parallelization. If `paral = TRUE` and `threads = NULL` then `threads` is set to a half of a maximum number of available threads

Value

A list containing the location of the matrix multiplication result:

fn Character string. Path to the HDF5 file containing the result

ds Character string. Full dataset path to the A*B multiplication result within the HDF5 file

Examples

```
library("BigDataStatMeth")
library("rhdf5")

N = 1000; M = 1000

set.seed(555)
a <- matrix( rnorm( N*M, mean=0, sd=1), N, M)
b <- matrix( rnorm( N*M, mean=0, sd=1), M, N)

fn <- "test_temp.hdf5"
bdCreate_hdf5_matrix(filename = fn,
                     object = a, group = "groupA",
                     dataset = "datasetA",
                     transp = FALSE,
                     overwriteFile = TRUE,
                     overwriteDataset = FALSE,
                     unlimited = FALSE)

bdCreate_hdf5_matrix(filename = fn,
                     object = t(b),
                     group = "groupA",
                     dataset = "datasetB",
                     transp = FALSE,
                     overwriteFile = FALSE,
                     overwriteDataset = TRUE,
                     unlimited = FALSE)

# Multiply two matrix
res <- bdblockmult_hdf5(filename = fn, group = "groupA",
    A = "datasetA", B = "datasetB", outgroup = "results",
    outdataset = "res", overwrite = TRUE )

# list contents
h5ls(fn)

# Extract the result from HDF5
result_hdf5 <- h5read(res$fn, res$ds)[1:3, 1:5]
result_hdf5

# Compute the same multiplication in R
result_r <- (a %*% b)[1:3, 1:5]
result_r

# Compare both results (should be TRUE)
all.equal(result_hdf5, result_r)

# Remove file
if (file.exists(fn)) {
  file.remove(fn)
}
```

bdblockmult_sparse_hdf5

Block matrix multiplication for sparse matrices

Description

Performs optimized block-wise matrix multiplication for sparse matrices stored in HDF5 format. The implementation is specifically designed to handle large sparse matrices efficiently through block operations and parallel processing.

Usage

```
bdblockmult_sparse_hdf5(
  filename,
  group,
  A,
  B,
  groupB = NULL,
  block_size = NULL,
  mixblock_size = NULL,
  paral = NULL,
  threads = NULL,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String indicating the HDF5 file path
`group`	String indicating the group path for matrix A
`A`	String specifying the dataset name for matrix A
`B`	String specifying the dataset name for matrix B
`groupB`	Optional string indicating group path for matrix B. If NULL, uses same group as A
`block_size`	Optional integer specifying block size for processing. If NULL, automatically determined based on matrix dimensions
`mixblock_size`	Optional integer for memory block size in parallel processing
`paral`	Optional boolean indicating whether to use parallel processing. Default is false
`threads`	Optional integer specifying number of threads for parallel processing. If NULL, uses maximum available threads
`outgroup`	Optional string specifying output group. Default is "OUTPUT"
`outdataset`	Optional string specifying output dataset name. Default is "A_x_B"
`overwrite`	Optional boolean indicating whether to overwrite existing datasets. Default is false

Details

The function implements optimized sparse matrix multiplication through:

- Block-wise processing to manage memory usage
- Automatic block size optimization
- Parallel processing support
- Efficient sparse matrix storage

Block size optimization considers:

- Available system memory
- Matrix dimensions and sparsity
- Parallel processing requirements

Memory efficiency is achieved through:

- Sparse matrix storage format
- Block-wise processing
- Minimal temporary storage
- Proper resource cleanup

Value

Modifies the HDF5 file in place, adding the multiplication result

Examples

```
## Not run:
library(Matrix)
library(BigDataStatMeth)

# Create sparse test matrices
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)
)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", as.matrix(x_sparse), "SPARSE", "x_sparse")
bdCreate_hdf5_matrix("test.hdf5", as.matrix(y_sparse), "SPARSE", "y_sparse")
```

```
# Perform multiplication
bdblockmult_sparse_hdf5("test.hdf5", "SPARSE", "x_sparse", "y_sparse",
                        block_size = 1024,
                        paral = TRUE,
                        threads = 4)

## End(Not run)
```

---

bdblockSubstract *Block-Based Matrix Subtraction*

Description

Performs efficient matrix subtraction using block-based algorithms. The function supports various input combinations (matrix-matrix, matrix-vector, vector-vector) and provides options for parallel processing and block-based computation.

Usage

```
bdblockSubstract(
  A,
  B,
  block_size = NULL,
  paral = NULL,
  byBlocks = TRUE,
  threads = NULL
)
```

Arguments

`A`	Matrix or vector. First input operand.
`B`	Matrix or vector. Second input operand.
`block_size`	Integer. Block size for computation. If NULL, uses maximum allowed block size.
`paral`	Logical. If TRUE, enables parallel computation. Default is FALSE.
`byBlocks`	Logical. If TRUE (default), forces block-based computation for large matrices. Can be set to FALSE to disable blocking.
`threads`	Integer. Number of threads for parallel computation. If NULL, uses half of available threads.

Details

This function implements block-based matrix subtraction algorithms optimized for cache efficiency and memory usage. Key features:

- Input combinations supported:
  - Matrix-matrix subtraction
  - Matrix-vector subtraction (both left and right)
  - Vector-vector subtraction
- Performance optimizations:
  - Block-based computation for cache efficiency
  - Parallel processing for large matrices
  - Automatic method selection based on input size
  - Memory-efficient implementation

The function automatically selects the appropriate subtraction method based on input types and sizes. For large matrices (>2.25e+08 elements), block-based computation is used by default.

Value

Matrix or vector containing the result of A - B.

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- Kumar, V. et al. (1994). Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publishing Company.

See Also

- `bdblockSum` for block-based matrix addition
- `bdblockMult` for block-based matrix multiplication

Examples

```
## Not run:
library(BigDataStatMeth)

# Matrix-matrix subtraction
N <- 2500
M <- 400
nc <- 4

set.seed(555)
mat1 <- matrix(rnorm(N*M, mean=0, sd=10), N, M)
mat2 <- matrix(rnorm(N*M, mean=0, sd=10), N, M)

# Parallel block subtraction
```

```
result <- bdblockSubstract(mat1, mat2,
                          paral = TRUE,
                          threads = nc)

# Matrix-vector subtraction
vec <- rnorm(M)
result_mv <- bdblockSubstract(mat1, vec,
                             paral = TRUE,
                             threads = nc)

## End(Not run)
```

bdblockSubstract_hdf5 *HDF5 dataset subtraction*

Description

Performs optimized block-wise subtraction between two datasets stored in HDF5 format. Supports both matrix-matrix and matrix-vector operations with memory-efficient block processing.

Usage

```
bdblockSubstract_hdf5(
  filename,
  group,
  A,
  B,
  groupB = NULL,
  block_size = NULL,
  paral = NULL,
  threads = NULL,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String indicating the HDF5 file path
`group`	String indicating the group containing matrix A
`A`	String specifying the dataset name for matrix A
`B`	String specifying the dataset name for matrix B
`groupB`	Optional string indicating group containing matrix B. If NULL, uses same group as A
`block_size`	Optional integer specifying block size for processing. If NULL, automatically determined based on matrix dimensions

`paral`	Optional boolean indicating whether to use parallel processing. Default is false
`threads`	Optional integer specifying number of threads for parallel processing. If NULL, uses maximum available threads
`outgroup`	Optional string specifying output group. Default is "OUTPUT"
`outdataset`	Optional string specifying output dataset name. Default is "A_-_B"
`overwrite`	Optional boolean indicating whether to overwrite existing datasets. Default is false

Details

The function implements optimized subtraction through:

Operation modes:

- Matrix-matrix subtraction (A - B)
- Matrix-vector subtraction
- Vector-matrix subtraction

Block processing:

- Automatic block size selection
- Memory-efficient operations
- Parallel computation support

Block size optimization based on:

- Matrix dimensions
- Available memory
- Operation type (matrix/vector)

Error handling:

- Dimension validation
- Resource management
- Exception handling

Value

A list containing the location of the subtraction result:

fn Character string. Path to the HDF5 file containing the result

ds Character string. Full dataset path to the subtraction result (A - B) within the HDF5 file

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
N <- 1500
M <- 1500
set.seed(555)
a <- matrix(rnorm(N*M), N, M)
b <- matrix(rnorm(N*M), N, M)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", a, "data", "A",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix("test.hdf5", b, "data", "B",
                     overwriteFile = FALSE)

# Perform subtraction
bdblockSubstract_hdf5("test.hdf5", "data", "A", "B",
                      outgroup = "results",
                      outdataset = "diff",
                      block_size = 1024,
                      paral = TRUE)

## End(Not run)
```

---

bdblockSum *Block-Based Matrix Addition*

Description

Performs efficient matrix addition using block-based algorithms. The function supports various input combinations (matrix-matrix, matrix-vector, vector-vector) and provides options for parallel processing and block-based computation.

Usage

```
bdblockSum(
  A,
  B,
  block_size = NULL,
  paral = NULL,
  byBlocks = TRUE,
  threads = NULL
)
```

Arguments

`A`	Matrix or vector. First input operand.
`B`	Matrix or vector. Second input operand.
`block_size`	Integer. Block size for computation. If NULL, uses maximum allowed block size.
`paral`	Logical. If TRUE, enables parallel computation. Default is FALSE.
`byBlocks`	Logical. If TRUE (default), forces block-based computation for large matrices. Can be set to FALSE to disable blocking.
`threads`	Integer. Number of threads for parallel computation. If NULL, uses half of available threads.

Details

This function implements block-based matrix addition algorithms optimized for cache efficiency and memory usage. Key features:

- Input combinations supported:
  - Matrix-matrix addition
  - Matrix-vector addition (both left and right)
  - Vector-vector addition
- Performance optimizations:
  - Block-based computation for cache efficiency
  - Parallel processing for large matrices
  - Automatic method selection based on input size
  - Memory-efficient implementation

The function automatically selects the appropriate addition method based on input types and sizes. For large matrices (>2.25e+08 elements), block-based computation is used by default.

Value

Matrix or vector containing the result of A + B.

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- Kumar, V. et al. (1994). Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publishing Company.

See Also

- `bdblockSubstract` for block-based matrix subtraction
- `bdblockMult` for block-based matrix multiplication

Examples

```
## Not run:
library(BigDataStatMeth)

# Matrix-matrix addition
N <- 2500
M <- 400
nc <- 4

set.seed(555)
mat1 <- matrix(rnorm(N*M, mean=0, sd=10), N, M)
mat2 <- matrix(rnorm(N*M, mean=0, sd=10), N, M)

# Parallel block addition
result <- bdblockSum(mat1, mat2,
                     paral = TRUE,
                     threads = nc)

# Matrix-vector addition
vec <- rnorm(M)
result_mv <- bdblockSum(mat1, vec,
                        paral = TRUE,
                        threads = nc)

## End(Not run)
```

`bdblockSum_hdf5` *HDF5 dataset addition*

Description

Performs optimized block-wise addition between two datasets stored in HDF5 format. Supports both matrix-matrix and matrix-vector operations with memory-efficient block processing.

Usage

```
bdblockSum_hdf5(
  filename,
  group,
  A,
  B,
  groupB = NULL,
  block_size = NULL,
  paral = NULL,
  threads = NULL,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String indicating the HDF5 file path
`group`	String indicating the group containing matrix A
`A`	String specifying the dataset name for matrix A
`B`	String specifying the dataset name for matrix B
`groupB`	Optional string indicating group containing matrix B. If NULL, uses same group as A
`block_size`	Optional integer specifying block size for processing. If NULL, automatically determined based on matrix dimensions
`paral`	Optional boolean indicating whether to use parallel processing. Default is false
`threads`	Optional integer specifying number of threads for parallel processing. If NULL, uses maximum available threads
`outgroup`	Optional string specifying output group. Default is "OUTPUT"
`outdataset`	Optional string specifying output dataset name. Default is "A_+_B"
`overwrite`	Optional boolean indicating whether to overwrite existing datasets. Default is false

Details

The function implements optimized addition through:

Operation modes:

- Matrix-matrix addition (A + B)
- Matrix-vector addition
- Vector-matrix addition

Block processing:

- Automatic block size selection
- Memory-efficient operations
- Parallel computation support

Block size optimization based on:

- Matrix dimensions
- Available memory
- Operation type (matrix/vector)

Error handling:

- Dimension validation
- Resource management
- Exception handling

Value

A list containing the location of the addition result:

fn Character string. Path to the HDF5 file containing the result

ds Character string. Full dataset path to the addition result (A + B) within the HDF5 file

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
N <- 1500
M <- 1500
set.seed(555)
a <- matrix(rnorm(N*M), N, M)
b <- matrix(rnorm(N*M), N, M)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", a, "data", "A",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix("test.hdf5", b, "data", "B",
                     overwriteFile = FALSE)

# Perform addition
bdblockSum_hdf5("test.hdf5", "data", "A", "B",
                outgroup = "results",
                outdataset = "sum",
                block_size = 1024,
                paral = TRUE)

## End(Not run)
```

---

bdCheckMatrix_hdf5 *Check Matrix Suitability for Eigenvalue Decomposition with Spectra*

Description

Checks whether a matrix stored in HDF5 format is suitable for eigenvalue decomposition using Spectra. The function verifies that the matrix is square and optionally checks for symmetry to recommend the best solver type.

Usage

```
bdCheckMatrix_hdf5(
  filename,
  group = NULL,
  dataset = NULL,
  check_symmetry = NULL,
  tolerance = NULL,
  sample_size = NULL
)
```

Arguments

`filename`	Character string. Path to the HDF5 file containing the matrix.
`group`	Character string. Path to the group containing the dataset.
`dataset`	Character string. Name of the dataset to check.
`check_symmetry`	Logical. Whether to check if the matrix is symmetric (default = TRUE).
`tolerance`	Numeric. Tolerance for symmetry checking (default = 1e-12).
`sample_size`	Integer. Number of elements to sample for large matrices (default = 1000).

Value

A list with matrix properties and suitability assessment.

Examples

```
## Not run:
# Check matrix suitability
check_result <- bdEigen_check_matrix("data.h5", "matrices", "my_matrix")

if (check_result$suitable_for_eigen) {
  # Use appropriate solver based on recommendation
  if (check_result$recommended_solver == "symmetric") {
    result <- bdEigen_hdf5("data.h5", "matrices", "my_matrix", which = "LA")
  } else {
    result <- bdEigen_hdf5("data.h5", "matrices", "my_matrix", which = "LM")
  }
} else {
  cat("Matrix is not suitable for eigendecomposition\n")
}

## End(Not run)
```

`bdCholesky_hdf5` *Cholesky Decomposition for HDF5-Stored Matrices*

Description

Computes the Cholesky decomposition of a symmetric positive-definite matrix stored in an HDF5 file. The Cholesky decomposition factors a matrix A into the product A = LL' where L is a lower triangular matrix.

Usage

```
bdCholesky_hdf5(
  filename,
  group,
  dataset,
  outdataset,
  outgroup = NULL,
  fullMatrix = NULL,
  overwrite = NULL,
  threads = NULL,
  elementsBlock = 1000000L
)
```

Arguments

`filename`	Character string. Path to the HDF5 file containing the input matrix.
`group`	Character string. Path to the group containing the input dataset.
`dataset`	Character string. Name of the input dataset to decompose.
`outdataset`	Character string. Name for the output dataset.
`outgroup`	Character string. Optional output group path. If not provided, results are stored in the input group.
`fullMatrix`	Logical. If TRUE, stores the complete matrix. If FALSE (default), stores only the lower triangular part to save space.
`overwrite`	Logical. If TRUE, allows overwriting existing results.
`threads`	Integer. Number of threads for parallel computation.
`elementsBlock`	Integer. Maximum number of elements to process in each block (default = 100,000). For matrices larger than 5000x5000, automatically adjusted to number of rows or columns * 2.

Details

The Cholesky decomposition is a specialized factorization for symmetric positive-definite matrices that provides several advantages:

- More efficient than LU decomposition for symmetric positive-definite matrices
- Numerically stable
- Useful for solving linear systems and computing matrix inverses
- Important in statistical computing (e.g., for sampling from multivariate normal distributions)

This implementation features:

- Block-based computation for large matrices
- Optional storage formats (full or triangular)
- Parallel processing support
- Memory-efficient block algorithm

Mathematical Details: For a symmetric positive-definite matrix A, the decomposition A = LL' has the following properties:

- L is lower triangular
- L has positive diagonal elements
- L is unique

The elements of L are computed using:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

$$l_{ji} = \frac{1}{l_{ii}}(a_{ji} - \sum_{k=1}^{i-1} l_{ik} l_{jk})$$

Value

A list containing the location of the Cholesky decomposition result:

fn Character string. Path to the HDF5 file containing the result

ds Character string. Full dataset path to the Cholesky decomposition result within the HDF5 file

L The lower triangular Cholesky factor

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- Higham, N. J. (2009). Cholesky factorization. Wiley Interdisciplinary Reviews: Computational Statistics, 1(2), 251-254.

See Also

- bdInvCholesky_hdf5 for computing inverse using Cholesky decomposition
- bdSolve_hdf5 for solving linear systems

Examples

```
## Not run:
library(rhdf5)

# Create a symmetric positive-definite matrix
set.seed(1234)
X <- matrix(rnorm(100), 10, 10)
A <- crossprod(X)  # A = X'X is symmetric positive-definite

# Save to HDF5
h5createFile("matrix.h5")
h5write(A, "matrix.h5", "data/matrix")
```

```
# Compute Cholesky decomposition
bdCholesky_hdf5("matrix.h5", "data", "matrix",
                outdataset = "chol",
                outgroup = "decompositions",
                fullMatrix = FALSE)

# Verify the decomposition
L <- h5read("matrix.h5", "decompositions/chol")
max(abs(A - L %*% t(L)))  # Should be very small

## End(Not run)
```

`bdcomputeMatrixVector_hdf5`

Apply Vector Operations to HDF5 Matrix

Description

Performs element-wise operations between a matrix and a vector stored in HDF5 format. The function supports addition, subtraction, multiplication, division and power operations, with options for row-wise or column-wise application and parallel processing.

Usage

```
bdcomputeMatrixVector_hdf5(
  filename,
  group,
  dataset,
  vectorgroup,
  vectordataset,
  outdataset,
  func,
  outgroup = NULL,
  byrows = NULL,
  paral = NULL,
  threads = NULL,
  overwrite = FALSE
)
```

Arguments

`filename`	String. Path to the HDF5 file containing the datasets.
`group`	String. Path to the group containing the matrix dataset.
`dataset`	String. Name of the matrix dataset.
`vectorgroup`	String. Path to the group containing the vector dataset.

`vectordataset`	String. Name of the vector dataset.
`outdataset`	String. Name for the output dataset.
`func`	String. Operation to perform: "+", "-", "*", "/", or "pow".
`outgroup`	Optional string. Output group path. If not provided, results are stored in the same group as the input matrix.
`byrows`	Logical. If TRUE, applies operation by rows. If FALSE (default), applies operation by columns.
`paral`	Logical. If TRUE, enables parallel processing.
`threads`	Integer. Number of threads for parallel processing. Ignored if paral is FALSE.
`overwrite`	Logical. If TRUE, allows overwriting existing datasets.

Details

This function provides a flexible interface for performing element-wise operations between matrices and vectors stored in HDF5 format. It supports:

- Four basic operations:
  - Addition (+): Adds vector elements to matrix rows/columns
  - Subtraction (-): Subtracts vector elements from matrix rows/columns
  - Multiplication (*): Multiplies matrix rows/columns by vector elements
  - Division (/): Divides matrix rows/columns by vector elements
  - Power (pow): power matrix rows/columns by vector elements
- Processing options:
  - Row-wise or column-wise operations
  - Parallel processing for improved performance
  - Configurable thread count for parallel execution
  - Memory-efficient processing for large datasets

The function performs extensive validation:

- Checks matrix and vector dimensions for compatibility
- Validates operation type
- Verifies HDF5 file and dataset accessibility
- Ensures proper data structures (matrix vs. vector)

Value

List with components:

fn Character string with the HDF5 filename

gr Character string with the HDF5 group

ds Character string with the full dataset path (group/dataset)

References

- The HDF Group. (2000-2010). HDF5 User's Guide.
- Eddelbuettel, D., & François, R. (2011). Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1-18.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
library(BigDataStatMeth)

# Create test data
set.seed(123)
Y <- matrix(rnorm(100), 10, 10)
X <- matrix(rnorm(10), 10, 1)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", Y, "data", "Y",
                     overwriteFile = TRUE,
                     overwriteDataset = FALSE,
                     unlimited = FALSE)
bdCreate_hdf5_matrix("test.hdf5", X, "data", "X",
                     overwriteFile = FALSE,
                     overwriteDataset = FALSE,
                     unlimited = FALSE)

# Multiply matrix rows by vector
bdcomputeMatrixVector_hdf5("test.hdf5",
                           group = "data",
                           dataset = "Y",
                           vectorgroup = "data",
                           vectordataset = "X",
                           outdataset = "ProdComputed",
                           func = "*",
                           byrows = TRUE,
                           overwrite = TRUE)

# Subtract vector from matrix rows
bdcomputeMatrixVector_hdf5("test.hdf5",
                           group = "data",
                           dataset = "Y",
                           vectorgroup = "data",
                           vectordataset = "X",
                           outdataset = "SubsComputed",
                           func = "-",
                           byrows = TRUE,
                           overwrite = TRUE)

# Subtract vector from matrix columns
```

```
bdcomputeMatrixVector_hdf5("test.hdf5",
                           group = "data",
                           dataset = "Y",
                           vectorgroup = "data",
                           vectordataset = "X",
                           outdataset = "SubsComputed",
                           func = "-",
                           byrows = FALSE,
                           overwrite = TRUE)

# Cleanup
if (file.exists("test.hdf5")) {
  file.remove("test.hdf5")
}
```

---

bdCorr_hdf5 — *Compute correlation matrix for matrices stored in HDF5 format*

Description

This function computes Pearson or Spearman correlation matrix for matrices stored in HDF5 format. It automatically detects whether to compute:

- Single matrix correlation cor(X) - when only dataset_x is provided
- Cross-matrix correlation cor(X,Y) - when both dataset_x and dataset_y are provided

It automatically selects between direct computation for small matrices and block-wise processing for large matrices to optimize memory usage and performance.

Correlation types supported:

- Single matrix: cor(X) when only dataset_x provided
- Single matrix transposed: cor(t(X)) when trans_x=TRUE
- Cross-correlation: cor(X,Y) when both datasets provided
- Cross with transpose: cor(t(X),Y), cor(X,t(Y)), cor(t(X),t(Y))

For omics data analysis:

- trans_x=FALSE, trans_y=FALSE: Variables vs Variables (genes vs genes, CpGs vs CpGs)
- trans_x=TRUE, trans_y=FALSE: Samples vs Variables (individuals vs genes)
- trans_x=FALSE, trans_y=TRUE: Variables vs Samples (genes vs individuals)
- trans_x=TRUE, trans_y=TRUE: Samples vs Samples (individuals vs individuals) - optimized to cor(X,Y)

Usage

```
bdCorr_hdf5(
  filename_x,
  group_x,
  dataset_x,
  filename_y = "",
  group_y = "",
  dataset_y = "",
  trans_x = FALSE,
  trans_y = FALSE,
  method = "pearson",
  use_complete_obs = TRUE,
  compute_pvalues = TRUE,
  block_size = 1000L,
  overwrite = FALSE,
  output_filename = "",
  output_group = "",
  output_dataset_corr = "",
  output_dataset_pval = "",
  threads = -1L
)
```

Arguments

`filename_x`	Character string with the path to the HDF5 file containing matrix X
`group_x`	Character string indicating the group containing matrix X
`dataset_x`	Character string indicating the dataset name of matrix X
`filename_y`	Character string with the path to the HDF5 file containing matrix Y (optional, default: "")
`group_y`	Character string indicating the group containing matrix Y (optional, default: "")
`dataset_y`	Character string indicating the dataset name of matrix Y (optional, default: "")
`trans_x`	Logical, whether to transpose matrix X (default: FALSE)
`trans_y`	Logical, whether to transpose matrix Y (default: FALSE, ignored for single matrix)
`method`	Character string indicating correlation method ("pearson" or "spearman", default: "pearson")
`use_complete_obs`	Logical, whether to use only complete observations (default: TRUE)
`compute_pvalues`	Logical, whether to compute p-values for correlations (default: TRUE)
`block_size`	Integer, block size for large matrix processing (default: 1000)
`overwrite`	Logical, whether to overwrite existing results (default: FALSE)
`output_filename`	Character string, output HDF5 file (default: same as filename_x)

`output_group`	Character string, custom output group name (default: auto-generated)
`output_dataset_corr`	Character string, custom correlation dataset name (default: "correlation")
`output_dataset_pval`	Character string, custom p-values dataset name (default: "pvalues")
`threads`	Integer, number of threads for parallel computation (optional, default: auto)

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the correlation matrix (group/dataset)

Examples

```
## Not run:
# Backward compatible - existing code works unchanged
result_original <- bdCorr_hdf5("data.h5", "expression", "genes")

# New transpose functionality
# Gene-gene correlations (variables)
gene_corr <- bdCorr_hdf5("omics.h5", "expression", "genes", trans_x = FALSE)

# Sample-sample correlations (individuals)
sample_corr <- bdCorr_hdf5("omics.h5", "expression", "genes", trans_x = TRUE)

# Cross-correlation: genes vs methylation sites (variables vs variables)
cross_vars <- bdCorr_hdf5("omics.h5", "expression", "genes",
                          "omics.h5", "methylation", "cpg_sites",
                          trans_x = FALSE, trans_y = FALSE)

# Cross-correlation: samples vs methylation sites (samples vs variables)
samples_vs_cpg <- bdCorr_hdf5("omics.h5", "expression", "genes",
                              "omics.h5", "methylation", "cpg_sites",
                              trans_x = TRUE, trans_y = FALSE)

## End(Not run)
```

`bdCorr_matrix` *Compute correlation matrix for in-memory matrices (unified function)*

Description

Compute Pearson or Spearman correlation matrix for matrices that fit in memory. This function automatically detects whether to compute:

- Single matrix correlation cor(X) - when only matrix X is provided
- Cross-correlation cor(X,Y) - when both matrices X and Y are provided

Usage

```
bdCorr_matrix(
  X,
  Y = NULL,
  trans_x = NULL,
  trans_y = NULL,
  method = NULL,
  use_complete_obs = NULL,
  compute_pvalues = NULL,
  threads = NULL
)
```

Arguments

X	First numeric matrix (observations in rows, variables in columns)
Y	Second numeric matrix (optional, observations in rows, variables in columns)
trans_x	Logical, whether to transpose matrix X (default: FALSE)
trans_y	Logical, whether to transpose matrix Y (default: FALSE, ignored if Y not provided)
method	Character string indicating correlation method ("pearson" or "spearman", default: "pearson")
use_complete_obs	Logical, whether to use only complete observations (default: TRUE)
compute_pvalues	Logical, whether to compute p-values for correlations (default: TRUE)
threads	Integer, number of threads for parallel computation (optional, default: -1 for auto)

Value

A list containing correlation results

Examples

```
## Not run:
# Backward compatible - existing code unchanged
set.seed(123)
X <- matrix(rnorm(1000), ncol = 10)
result_original <- bdCorr_matrix(X)

# Create omics-style data
gene_expr <- matrix(rnorm(5000), nrow = 100, ncol = 50)  # 100 samples × 50 genes

# Gene-gene correlations (variables)
gene_corr <- bdCorr_matrix(gene_expr, trans_x = FALSE)

# Sample-sample correlations (individuals)
sample_corr <- bdCorr_matrix(gene_expr, trans_x = TRUE)
```

```
# Cross-correlation examples
methylation <- matrix(rnorm(4000), nrow = 100, ncol = 40)  # 100 samples × 40 CpGs

# Variables vs variables (genes vs CpGs)
vars_vs_vars <- bdCorr_matrix(gene_expr, methylation,
                              trans_x = FALSE, trans_y = FALSE)

# Samples vs variables (individuals vs CpGs)
samples_vs_vars <- bdCorr_matrix(gene_expr, methylation,
                                 trans_x = TRUE, trans_y = FALSE)

## End(Not run)
```

---

bdCreate_diagonal_hdf5 *Create Diagonal Matrix or Vector in HDF5 File*

Description

Creates a diagonal matrix or vector directly in an HDF5 file using block-wise processing to minimize memory usage. This unified function replaces separate diagonal and identity matrix creation functions, providing flexible diagonal creation with automatic parameter detection.

Usage

```
bdCreate_diagonal_hdf5(
  filename,
  group,
  dataset,
  size = NULL,
  scalar = 1,
  diagonal_values = NULL,
  output_type = "matrix",
  block_size = 0L,
  compression = 6L,
  overwriteFile = NULL,
  overwriteDataset = NULL,
  threads = NULL
)
```

Arguments

`filename`	Character. Path to HDF5 file
`group`	Character. Group path in HDF5 file (default: "/")
`dataset`	Character. Name of dataset to create

`size`	Integer. Size of diagonal (auto-detected if diagonal_values provided)
`scalar`	Numeric. Scalar multiplier for diagonal elements (default: 1.0)
`diagonal_values`	Numeric vector. Custom diagonal values (optional)
`output_type`	Character. Output format: "matrix" or "vector" (default: "matrix")
`block_size`	Integer. Block size for processing (default: auto-estimate)
`compression`	Integer. Compression level 0-9 (default: 6)
`overwriteFile`	Logical. Overwrite file if exists (default: FALSE)
`overwriteDataset`	Logical. Overwrite dataset if exists (default: FALSE)
`threads`	Integer. Number of threads to use (default: auto-detect)

Details

This function provides flexible diagonal creation with two main modes:

- Vector mode: Provide custom diagonal values
  - Size is automatically detected from vector length
  - Scalar acts as additional multiplier
  - Ideal for custom diagonal patterns
- Scalar mode: Provide size and scalar value
  - Creates uniform diagonal with specified scalar
  - scalar=1.0 creates identity matrix/vector
  - Ideal for identity or uniform diagonal matrices
- Output formats:
  - "matrix": Creates full N×N matrix (sparse, only diagonal populated)
  - "vector": Creates efficient 1×N vector with diagonal values only
- Performance features:
  - Block-wise processing for memory efficiency
  - Optional compression with configurable levels
  - Parallel processing support for large datasets
  - Automatic block size optimization

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the diagonal matrix (group/dataset)

Examples

```
## Not run:
library(BigDataStatMeth)

# Create identity matrix (1M x 1M)
bdCreate_diagonal_hdf5("identity.h5", "/", "I_matrix",
                      size = 1000000, scalar = 1.0)

# Create scaled identity vector (more efficient)
bdCreate_diagonal_hdf5("scaled_id.h5", "/", "scaled_I",
                      size = 500000, scalar = 3.14,
                      output_type = "vector")

# Create custom diagonal matrix
custom_diag <- runif(10000)
bdCreate_diagonal_hdf5("custom.h5", "/", "my_diag",
                      diagonal_values = custom_diag,
                      scalar = 2.0, output_type = "matrix")

# Create custom diagonal vector (most efficient)
bdCreate_diagonal_hdf5("custom_vec.h5", "/", "my_diag_vec",
                      diagonal_values = custom_diag,
                      output_type = "vector")

## End(Not run)
```

`bdCreate_hdf5_emptyDataset`

Create an empty HDF5 dataset (no data written)

Description

Creates an HDF5 dataset of size `nrows` × `ncols` inside `group` with name `dataset`, without writing data (allocation only). Honors file/dataset overwrite flags and supports unlimited datasets.

Usage

```
bdCreate_hdf5_emptyDataset(
  filename,
  group,
  dataset,
  nrows = 0L,
  ncols = 0L,
  overwriteFile = NULL,
  overwriteDataset = NULL,
  unlimited = NULL,
  datatype = NULL
)
```

Arguments

`filename`	Character. Path to the HDF5 file.
`group`	Character. Group path.
`dataset`	Character. Dataset name.
`nrows`	Integer (>= 1). Number of rows.
`ncols`	Integer (>= 1). Number of columns.
`overwriteFile`	Logical. If `TRUE`, allow file recreate default value `FALSE`.
`overwriteDataset`	Logical. If `TRUE`, replace dataset default value `FALSE`.
`unlimited`	Logical. If `TRUE`, create unlimited dataset default value `FALSE`.
`datatype`	Character. Element type (e.g., "real").

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the empty dataset (group/dataset)

Examples

```
## Not run:
bdCreate_hdf5_emptyDataset("test.h5", "MGCCA_IN", "X", 1000, 500,
                           overwriteFile = FALSE,
                           overwriteDataset = TRUE,
                           unlimited = FALSE,
                           datatype = "real")

## End(Not run)
```

`bdCreate_hdf5_group` *Create Group in an HDF5 File*

Description

Create a (nested) group inside an HDF5 file. The operation is idempotent: if the group already exists, no error is raised.

Usage

```
bdCreate_hdf5_group(filename, group)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Group path to create (e.g., `"MGCCA_OUT/scores"`).

Details

Intermediate groups are created when needed. The HDF5 file must exist prior to the call (create it with a writer function).

Value

List with components:

fn Character string with the HDF5 filename

gr Character string with the full group path created within the HDF5 file

References

The HDF Group. HDF5 User's Guide.

See Also

`bdCreate_hdf5_matrix`, `bdRemove_hdf5_element`

Examples

```
## Not run:
library(BigDataStatMeth)
fn <- "test.hdf5"

# Ensure file exists (e.g., by creating an empty dataset or via a helper)
mat <- matrix(0, nrow = 1, ncol = 1)
bdCreate_hdf5_matrix(fn, mat, group = "tmp", dataset = "seed",
                     overwriteFile = TRUE)

# Create nested group
bdCreate_hdf5_group(fn, "MGCCA_OUT/scores")

## End(Not run)
```

bdCreate_hdf5_matrix *Create hdf5 data file and write data to it*

Description

Creates a hdf5 file with numerical data matrix,

Usage

```
bdCreate_hdf5_matrix(
  filename,
  object,
  group = NULL,
  dataset = NULL,
  transp = NULL,
  overwriteFile = NULL,
  overwriteDataset = NULL,
  unlimited = 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
`overwriteFile`	optional boolean by default overwriteFile = false, if true and file exists, removes old file and creates a new file with de dataset data.
`overwriteDataset`	optional boolean by default overwriteDataset = false, if true and dataset exists, removes old dataset and creates a new dataset.
`unlimited`	optional boolean by default unlimited = false, if true creates a dataset that can growth.

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the created matrix (group/dataset)

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(filename = "test_temp.hdf5",
                    object = matA, group = "datasets",
                    dataset = "datasetA", transp = FALSE,
                    overwriteFile = TRUE,
                    overwriteDataset = TRUE,
                    unlimited = FALSE)

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

`bdCrossprod` *Efficient Matrix Cross-Product Computation*

Description

Computes matrix cross-products efficiently using block-based algorithms and optional parallel processing. Supports both single-matrix (X'X) and two-matrix (X'Y) cross-products.

Usage

```
bdCrossprod(
  A,
  B = NULL,
  transposed = NULL,
  block_size = NULL,
  paral = NULL,
  threads = NULL
)
```

Arguments

`A`	Numeric matrix. First input matrix.
`B`	Optional numeric matrix. If provided, computes A'B instead of A'A.
`transposed`	Logical. If TRUE, uses transposed input matrix.
`block_size`	Integer. Block size for computation. If NULL, uses optimal block size based on matrix dimensions and cache size.
`paral`	Logical. If TRUE, enables parallel computation.
`threads`	Integer. Number of threads for parallel computation. If NULL, uses all available threads.

Details

This function implements efficient cross-product computation using block-based algorithms optimized for cache efficiency and memory usage. Key features:

- Operation modes:
  - Single matrix: Computes X'X
  - Two matrices: Computes X'Y
- Performance optimizations:
  - Block-based computation for cache efficiency
  - Parallel processing for large matrices
  - Automatic block size selection
  - Memory-efficient implementation

The function automatically selects optimal computation strategies based on input size and available resources. For large matrices, block-based computation is used to improve cache utilization.

Value

Numeric matrix containing the cross-product result.

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- Kumar, V. et al. (1994). Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publishing Company.

See Also

- `bdtCrossprod` for transposed cross-product
- `bdblockMult` for block-based matrix multiplication

Examples

```
library(BigDataStatMeth)

# Single matrix cross-product
n <- 100
p <- 60
X <- matrix(rnorm(n*p), nrow=n, ncol=p)
res <- bdCrossprod(X)

# Verify against base R
all.equal(crossprod(X), res)

# Two-matrix cross-product
n <- 100
p <- 100
Y <- matrix(rnorm(n*p), nrow=n)
```

```
res <- bdCrossprod(X, Y)

# Parallel computation
res_par <- bdCrossprod(X, Y,
                       paral = TRUE,
                       threads = 4)
```

bdCrossprod_hdf5 *Crossprod with hdf5 matrix*

Description

Performs optimized cross product operations on matrices stored in HDF5 format. For a single matrix A, computes A^t * A. For two matrices A and B, computes A^t * B. Uses block-wise processing for memory efficiency.

Usage

```
bdCrossprod_hdf5(
  filename,
  group,
  A,
  B = NULL,
  groupB = NULL,
  block_size = NULL,
  mixblock_size = NULL,
  paral = NULL,
  threads = NULL,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String indicating the HDF5 file path
`group`	String indicating the input group containing matrix A
`A`	String specifying the dataset name for matrix A
`B`	Optional string specifying dataset name for matrix B. If NULL, performs A^t * A
`groupB`	Optional string indicating group containing matrix B. If NULL, uses same group as A
`block_size`	Optional integer specifying the block size for processing. Default is automatically determined based on matrix dimensions
`mixblock_size`	Optional integer for memory block size in parallel processing

`paral`	Optional boolean indicating whether to use parallel processing. Default is false
`threads`	Optional integer specifying number of threads for parallel processing. If NULL, uses maximum available threads
`outgroup`	Optional string specifying output group. Default is "OUTPUT"
`outdataset`	Optional string specifying output dataset name. Default is "CrossProd_A_x_B"
`overwrite`	Optional boolean indicating whether to overwrite existing datasets. Default is false

Details

The function implements block-wise matrix multiplication to handle large matrices efficiently. Block size is automatically optimized based on:

- Available memory
- Matrix dimensions
- Whether parallel processing is enabled

For parallel processing:

- Uses OpenMP for thread management
- Implements cache-friendly block operations
- Provides automatic thread count optimization

Memory efficiency is achieved through:

- Block-wise reading and writing
- Minimal temporary storage
- Proper resource cleanup

Value

A list containing the location of the crossproduct result:

fn Character string. Path to the HDF5 file containing the result

ds Character string. Full dataset path to the crossproduct result (t(A) *%% A or t(A) %%* B) within the HDF5 file

Examples

```
## Not run:
  library(BigDataStatMeth)
  library(rhdf5)

  # Create test matrix
  N = 1000
  M = 1000
  set.seed(555)
  a <- matrix(rnorm(N*M), N, M)
```

```
    # Save to HDF5
    bdCreate_hdf5_matrix("test.hdf5", a, "INPUT", "A", overwriteFile = TRUE)

    # Compute cross product
    bdCrossprod_hdf5("test.hdf5", "INPUT", "A",
                     outgroup = "OUTPUT",
                     outdataset = "result",
                     block_size = 1024,
                     paral = TRUE,
                     threads = 4)

## End(Not run)
```

bdDiag_add_hdf5 *Add Diagonal Elements from HDF5 Matrices or Vectors*

Description

Performs optimized diagonal addition between two datasets stored in HDF5 format. Automatically detects whether inputs are matrices (extracts diagonals) or vectors (direct operation) and uses the most efficient approach.

Usage

```
bdDiag_add_hdf5(
  filename,
  group,
  A,
  B,
  groupB = NULL,
  target = NULL,
  outgroup = NULL,
  outdataset = NULL,
  paral = NULL,
  threads = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String. Path to the HDF5 file containing the datasets.
`group`	String. Group path containing the first dataset (A).
`A`	String. Name of the first dataset (matrix or vector).
`B`	String. Name of the second dataset (matrix or vector).
`groupB`	Optional string. Group path containing dataset B.
`target`	Optional string. Where to write result: "A", "B", or "new" (default: "new").

`outgroup`	Optional string. Output group path (only used if target="new").
`outdataset`	Optional string. Output dataset name (only used if target="new").
`paral`	Optional logical. Whether to use parallel processing.
`threads`	Optional integer. Number of threads for parallel processing.
`overwrite`	Optional logical. Whether to overwrite existing datasets.

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the diagonal addition result (group/dataset)

`bdDiag_divide_hdf5` *Divide Diagonal Elements from HDF5 Matrices or Vectors*

Description

Performs optimized diagonal division between two datasets stored in HDF5 format. Automatically detects whether inputs are matrices (extracts diagonals) or vectors (direct operation) and uses the most efficient approach. This function is ~50-250x faster than traditional matrix operations for diagonal computations.

Usage

```
bdDiag_divide_hdf5(
  filename,
  group,
  A,
  B,
  groupB = NULL,
  target = NULL,
  outgroup = NULL,
  outdataset = NULL,
  paral = NULL,
  threads = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String. Path to the HDF5 file containing the datasets.
`group`	String. Group path containing the first dataset (A, dividend).
`A`	String. Name of the first dataset (dividend).
`B`	String. Name of the second dataset (divisor).

`groupB`	Optional string. Group path containing dataset B. If NULL, uses same group as A.
`target`	Optional string. Where to write result: "A", "B", or "new" (default: "new").
`outgroup`	Optional string. Output group path. Default is "OUTPUT".
`outdataset`	Optional string. Output dataset name. Default is "A_/_B" with .diag suffix if appropriate.
`paral`	Optional logical. Whether to use parallel processing. Default is FALSE.
`threads`	Optional integer. Number of threads for parallel processing. If NULL, uses maximum available threads.
`overwrite`	Optional logical. Whether to overwrite existing datasets. Default is FALSE.

Details

This function provides flexible diagonal division with automatic optimization:

- Operation modes:
  - Matrix / Matrix: Extract diagonals $\rightarrow$ vector division $\rightarrow$ save as vector
  - Matrix / Vector: Extract diagonal $\rightarrow$ vector division $\rightarrow$ save as vector
  - Vector / Vector: Direct vector division (most efficient)
- Performance features:
  - Uses optimized vector operations for maximum efficiency
  - Automatic type detection and dimension validation
  - Memory-efficient processing for large datasets
  - Parallel processing support for improved performance
- Mathematical properties:
  - Element-wise division: result[i] = A[i] / B[i]
  - Division by zero results in infinity (IEEE 754 standard)
  - Handles special cases: ±inf, NaN, and subnormal numbers
  - Order matters: $A/B \neq B/A$.

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the diagonal division result (group/dataset)

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
N <- 1000
set.seed(123)
```

```
A <- matrix(rnorm(N*N), N, N)
B <- matrix(rnorm(N*N, mean=1), N, N)  # Avoid division by zero

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", A, "data", "matrixA",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix("test.hdf5", B, "data", "matrixB",
                     overwriteFile = FALSE)

# Divide diagonals
result <- bdDiag_divide_hdf5("test.hdf5", "data", "matrixA", "matrixB",
                            outgroup = "results",
                            outdataset = "diagonal_ratio",
                            paral = TRUE)

## End(Not run)
```

bdDiag_multiply_hdf5 *Multiply Diagonal Elements from HDF5 Matrices or Vectors*

Description

Performs optimized diagonal multiplication between two datasets stored in HDF5 format. Automatically detects whether inputs are matrices (extracts diagonals) or vectors (direct operation) and uses the most efficient approach. This function performs element-wise multiplication and is ~50-250x faster than traditional matrix operations.

Usage

```
bdDiag_multiply_hdf5(
  filename,
  group,
  A,
  B,
  groupB = NULL,
  target = NULL,
  outgroup = NULL,
  outdataset = NULL,
  paral = NULL,
  threads = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String. Path to the HDF5 file containing the datasets.
`group`	String. Group path containing the first dataset (A).

`A`	String. Name of the first dataset (matrix or vector).
`B`	String. Name of the second dataset (matrix or vector).
`groupB`	Optional string. Group path containing dataset B. If NULL, uses same group as A.
`target`	Optional string. Where to write result: "A", "B", or "new" (default: "new").
`outgroup`	Optional string. Output group path. Default is "OUTPUT".
`outdataset`	Optional string. Output dataset name. Default is "A_*_B" with .diag suffix if appropriate.
`paral`	Optional logical. Whether to use parallel processing. Default is FALSE.
`threads`	Optional integer. Number of threads for parallel processing. If NULL, uses maximum available threads.
`overwrite`	Optional logical. Whether to overwrite existing datasets. Default is FALSE.

Details

This function provides flexible diagonal multiplication with automatic optimization:

- Operation modes:
  - Matrix * Matrix: Extract diagonals $\rightarrow$ vector multiplication $\rightarrow$ save as vector
  - Matrix * Vector: Extract diagonal $\rightarrow$ vector multiplication $\rightarrow$ save as vector
  - Vector * Vector: Direct vector multiplication (most efficient)
- Performance features:
  - Uses optimized vector operations for maximum efficiency
  - Automatic type detection and dimension validation
  - Memory-efficient processing for large datasets
  - Parallel processing support for improved performance
- Mathematical properties:
  - Element-wise multiplication (not matrix multiplication)
  - Commutative operation: A * B = B * A
  - Handles overflow according to IEEE 754 standards
  - Preserves sign information correctly

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the diagonal multiplication result (group/dataset)

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
N <- 1000
set.seed(123)
A <- matrix(rnorm(N*N), N, N)
B <- matrix(rnorm(N*N), N, N)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", A, "data", "matrixA",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix("test.hdf5", B, "data", "matrixB",
                     overwriteFile = FALSE)

# Multiply diagonals (element-wise)
result <- bdDiag_multiply_hdf5("test.hdf5", "data", "matrixA", "matrixB",
                              outgroup = "results",
                              outdataset = "diagonal_product",
                              paral = TRUE)

## End(Not run)
```

---

bdDiag_scalar_hdf5 *Apply Scalar Operations to Diagonal Elements*

Description

Performs optimized scalar operations on diagonal elements of matrices or vectors stored in HDF5 format. Automatically detects whether input is a matrix (extracts diagonal) or vector (direct operation) and applies the specified scalar operation.

Usage

```
bdDiag_scalar_hdf5(
  filename,
  group,
  dataset,
  scalar,
  operation,
  target = NULL,
  paral = NULL,
  threads = NULL,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String. Path to the HDF5 file containing the dataset.
`group`	String. Group path containing the input dataset.
`dataset`	String. Name of the input dataset (matrix or vector).
`scalar`	Numeric. Scalar value for the operation.
`operation`	String. Operation to perform: "add", "subtract", "multiply", "divide".
`target`	Optional string. Where to write result: "input" or "new" (default: "new").
`paral`	Optional logical. Whether to use parallel processing (default: FALSE).
`threads`	Optional integer. Number of threads for parallel processing.
`outgroup`	Optional string. Output group path (only used if target="new").
`outdataset`	Optional string. Output dataset name (only used if target="new").
`overwrite`	Optional logical. Whether to overwrite existing datasets (default: FALSE).

Details

This function provides flexible scalar operations on diagonals:

- Supported operations:
  - "+": diagonal[i] + scalar
  - "-": diagonal[i] - scalar
  - "*": diagonal[i] * scalar
  - "/": diagonal[i] / scalar
  - "pow": diagonal[i] ^ scalar
- Input types:
  - Matrix input: Extracts diagonal automatically
  - Vector input: Operates directly (most efficient)
- Target options:
  - "input": Modifies original dataset in-place
  - "new": Creates new dataset with result

Value

List with components:

fn Character string with the HDF5 filename

gr Character string with the HDF5 group

ds Character string with the full dataset path (group/dataset)

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrix
A <- matrix(rnorm(100), 10, 10)
bdCreate_hdf5_matrix("test.h5", A, "data", "matrix_A", overwriteFile = TRUE)

# Add scalar to diagonal (creates new dataset)
result <- bdDiag_scalar_hdf5("test.h5", "data", "matrix_A",
                             scalar = 5.0, operation = "+",
                             target = "new", outdataset = "diag_plus_5")

# Multiply diagonal in-place
result2 <- bdDiag_scalar_hdf5("test.h5", "data", "matrix_A",
                              scalar = 2.0, operation = "*",
                              target = "input")

## End(Not run)
```

---

bdDiag_subtract_hdf5 *Subtract Diagonal Elements from HDF5 Matrices or Vectors*

Description

Performs optimized diagonal subtraction between two datasets stored in HDF5 format. Automatically detects whether inputs are matrices (extracts diagonals) or vectors (direct operation) and uses the most efficient approach. This function is ~50-250x faster than traditional matrix operations for diagonal computations.

Usage

```
bdDiag_subtract_hdf5(
  filename,
  group,
  A,
  B,
  groupB = NULL,
  target = NULL,
  outgroup = NULL,
  outdataset = NULL,
  paral = NULL,
  threads = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String. Path to the HDF5 file containing the datasets.
`group`	String. Group path containing the first dataset (A, minuend).
`A`	String. Name of the first dataset (minuend).
`B`	String. Name of the second dataset (subtrahend).
`groupB`	Optional string. Group path containing dataset B. If NULL, uses same group as A.
`target`	Optional string. Where to write result: "A", "B", or "new" (default: "new").
`outgroup`	Optional string. Output group path. Default is "OUTPUT".
`outdataset`	Optional string. Output dataset name. Default is "A_-_B" with .diag suffix if appropriate.
`paral`	Optional logical. Whether to use parallel processing. Default is FALSE.
`threads`	Optional integer. Number of threads for parallel processing. If NULL, uses maximum available threads.
`overwrite`	Optional logical. Whether to overwrite existing datasets. Default is FALSE.

Details

This function provides flexible diagonal subtraction with automatic optimization:

- Operation modes:
  - Matrix - Matrix: Extract diagonals → vector subtraction → save as vector
  - Matrix - Vector: Extract diagonal → vector subtraction → save as vector
  - Vector - Vector: Direct vector subtraction (most efficient)
- Performance features:
  - Uses optimized vector operations for maximum efficiency
  - Automatic type detection and dimension validation
  - Memory-efficient processing for large datasets
  - Parallel processing support for improved performance
- Validation checks:
  - Matrix inputs must be square (N×N)
  - Vector inputs must have compatible dimensions
  - Automatic dimension matching between operands

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the diagonal subtraction result (group/dataset)

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
N <- 1000
set.seed(123)
A <- matrix(rnorm(N*N), N, N)
B <- matrix(rnorm(N*N), N, N)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", A, "data", "matrixA",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix("test.hdf5", B, "data", "matrixB",
                     overwriteFile = FALSE)

# Subtract diagonals
result <- bdDiag_subtract_hdf5("test.hdf5", "data", "matrixA", "matrixB",
                              outgroup = "results",
                              outdataset = "diagonal_diff",
                              paral = TRUE)

## End(Not run)
```

`bdEigen_hdf5` *Eigenvalue Decomposition for HDF5-Stored Matrices using Spectra*

Description

Computes the eigenvalue decomposition of a large matrix stored in an HDF5 file using the Spectra library. This provides consistent results with the RSpectra package and can handle both symmetric and non-symmetric matrices.

Usage

```
bdEigen_hdf5(
  filename,
  group = NULL,
  dataset = NULL,
  k = NULL,
  which = NULL,
  ncv = NULL,
  bcenter = NULL,
  bscale = NULL,
  tolerance = NULL,
  max_iter = NULL,
  compute_vectors = NULL,
  overwrite = NULL,
  threads = NULL
)
```

Arguments

`filename`	Character string. Path to the HDF5 file containing the input matrix.
`group`	Character string. Path to the group containing the input dataset.
`dataset`	Character string. Name of the input dataset to decompose.
`k`	Integer. Number of eigenvalues to compute (default = 6, following Spectra convention).
`which`	Character string. Which eigenvalues to compute (default = "LM"): • "LM": Largest magnitude • "SM": Smallest magnitude • "LR": Largest real part (non-symmetric matrices) • "SR": Smallest real part (non-symmetric matrices) • "LI": Largest imaginary part (non-symmetric matrices) • "SI": Smallest imaginary part (non-symmetric matrices) • "LA": Largest algebraic (symmetric matrices) • "SA": Smallest algebraic (symmetric matrices)
`ncv`	Integer. Number of Arnoldi vectors (default = 0, auto-selected as max(2*k+1, 20)).
`bcenter`	Logical. If TRUE, centers the data by subtracting column means (default = FALSE).
`bscale`	Logical. If TRUE, scales the centered columns by their standard deviations (default = FALSE).
`tolerance`	Numeric. Convergence tolerance for Spectra algorithms (default = 1e-10).
`max_iter`	Integer. Maximum number of iterations for Spectra algorithms (default = 1000).
`compute_vectors`	Logical. If TRUE (default), computes both eigenvalues and eigenvectors.
`overwrite`	Logical. If TRUE, allows overwriting existing results (default = FALSE).
`threads`	Integer. Number of threads for parallel computation (default = NULL, uses available cores).

Details

This function uses the Spectra library (same as RSpectra) for eigenvalue computation, ensuring consistent results. Key features include:

- Automatic detection of symmetric vs non-symmetric matrices
- Support for both real and complex eigenvalues/eigenvectors
- Memory-efficient block-based processing for large matrices
- Parallel processing support

- Various eigenvalue selection criteria
- Consistent interface with RSpectra::eigs()

The implementation automatically:

- Detects matrix symmetry and uses appropriate solver (SymEigsSolver vs GenEigsSolver)
- Handles complex eigenvalues for non-symmetric matrices
- Saves imaginary parts separately when non-zero
- Provides the same results as RSpectra::eigs() function

Value

List with components:

fn Character string with the HDF5 filename

values Character string with the full dataset path to the eigenvalues (real part) (group/dataset)

vectors Character string with the full dataset path to the eigenvectors (real part) (group/dataset)

values_imag Character string with the full dataset path to the eigenvalues (imaginary part), or NULL if all eigenvalues are real

vectors_imag Character string with the full dataset path to the eigenvectors (imaginary part), or NULL if all eigenvectors are real

is_symmetric Logical indicating whether the matrix was detected as symmetric

References

- Qiu, Y., & Mei, J. (2022). RSpectra: Solvers for Large-Scale Eigenvalue and SVD Problems.
- Li, R. (2021). Spectra: C++ Library For Large Scale Eigenvalue Problems.

See Also

- `bdSVD_hdf5` for Singular Value Decomposition
- `bdPCA_hdf5` for Principal Component Analysis
- `RSpectra::eigs` for the R equivalent function

Examples

```
## Not run:
library(BigDataStatMeth)
library(rhdf5)
library(RSpectra)

# Create a sample matrix (can be non-symmetric)
set.seed(123)
A <- matrix(rnorm(2500), 50, 50)

fn <- "test_eigen.hdf5"
bdCreate_hdf5_matrix_file(filename = fn, object = A, group = "data", dataset = "matrix")
```

```
# Compute eigendecomposition with BigDataStatMeth
res <- bdEigen_hdf5(fn, "data", "matrix", k = 6, which = "LM")

# Compare with RSpectra (should give same results)
rspectra_result <- eigs(A, k = 6, which = "LM")

# Extract results from HDF5
eigenvals_bd <- h5read(res$fn, res$values)
eigenvecs_bd <- h5read(res$fn, res$vectors)

# Compare eigenvalues (should be identical)
all.equal(eigenvals_bd, Re(rspectra_result$values), tolerance = 1e-12)

# For non-symmetric matrices, check imaginary parts
if (!is.null(res$values_imag)) {
  eigenvals_imag <- h5read(res$fn, res$values_imag)
  all.equal(eigenvals_imag, Im(rspectra_result$values), tolerance = 1e-12)
}

# Remove file
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

`bdgetDatasetsList_hdf5`

List Datasets in HDF5 Group

Description

Retrieves a list of all datasets within a specified HDF5 group, with optional filtering by prefix or suffix.

Usage

```
bdgetDatasetsList_hdf5(filename, group, prefix = NULL)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group within the HDF5 file.
`prefix`	Optional character string. If provided, only returns datasets starting with this prefix.

Details

This function provides flexible dataset listing capabilities for HDF5 files. Key features:

- Listing options:
  - All datasets in a group
  - Datasets matching a prefix
  - Datasets matching a suffix
- Implementation features:
  - Safe HDF5 file operations
  - Memory-efficient implementation
  - Comprehensive error handling
  - Read-only access to files

The function opens the HDF5 file in read-only mode to ensure data safety.

Value

Character vector containing dataset names.

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create a test HDF5 file
fn <- "test.hdf5"
X <- matrix(rnorm(100), 10, 10)
Y <- matrix(rnorm(100), 10, 10)

# Save matrices to HDF5
bdCreate_hdf5_matrix(fn, X, "data", "matrix1",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix(fn, Y, "data", "matrix2",
                     overwriteFile = FALSE)

# List all datasets in group
datasets <- bdgetDatasetsList_hdf5(fn, "data")
print(datasets)

# List datasets with prefix "matrix"
filtered <- bdgetDatasetsList_hdf5(fn, "data", prefix = "matrix")
print(filtered)
```

```
# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

`bdgetDiagonal_hdf5` *Get Matrix Diagonal from HDF5*

Description

Retrieves the diagonal elements from a matrix stored in an HDF5 file.

Usage

```
bdgetDiagonal_hdf5(filename, group, dataset)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing the dataset.
`dataset`	Character string. Name of the dataset.

Details

This function provides efficient access to matrix diagonal elements with:

- Access features:
  - Direct diagonal access
  - Memory-efficient retrieval
  - Support for large matrices
- Implementation features:
  - Safe HDF5 file operations
  - Memory-efficient implementation
  - Comprehensive error handling
  - Read-only access to files

The function opens the HDF5 file in read-only mode to ensure data safety.

Value

Numeric vector containing diagonal elements.

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdWriteDiagonal_hdf5` for writing diagonal elements
- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrix
X <- matrix(rnorm(100), 10, 10)
diag(X) <- 0.5

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", X, "data", "matrix1",
                     overwriteFile = TRUE)

# Get diagonal
diag_elements <- bdgetDiagonal_hdf5("test.hdf5", "data", "matrix1")
print(diag_elements)

# Cleanup
if (file.exists("test.hdf5")) {
  file.remove("test.hdf5")
}

## End(Not run)
```

`bdgetDim_hdf5` *Get HDF5 Dataset Dimensions*

Description

Retrieves the dimensions (number of rows and columns) of a dataset stored in an HDF5 file.

Usage

```
bdgetDim_hdf5(filename, dataset)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`dataset`	Character string. Full path to the dataset within the HDF5 file (e.g., "group/subgroup/dataset").

Details

This function provides efficient access to dataset dimensions in HDF5 files. Key features:

- Dimension information:
  - Number of rows
  - Number of columns
- Implementation features:
  - Safe HDF5 file operations
  - Memory-efficient implementation
  - Comprehensive error handling
  - Read-only access to files

The function opens the HDF5 file in read-only mode to ensure data safety.

Value

Integer vector of length 2 containing:

- [1] Number of rows
- [2] Number of columns

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdgetDatasetsList_hdf5` for listing available datasets
- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create a test HDF5 file
fn <- "test.hdf5"
X <- matrix(rnorm(100), 10, 10)

# Save matrix to HDF5
bdCreate_hdf5_matrix(fn, X, "data", "matrix1",
                     overwriteFile = TRUE)

# Get dimensions
dims <- bdgetDim_hdf5(fn, "data/matrix1")
print(paste("Rows:", dims[1]))
print(paste("Columns:", dims[2]))

# Cleanup
```

```
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

`bdgetSDandMean_hdf5` *Compute Matrix Standard Deviation and Mean in HDF5*

Description

Computes standard deviation and/or mean statistics for a matrix stored in HDF5 format, with support for row-wise or column-wise computations.

Usage

```
bdgetSDandMean_hdf5(
  filename,
  group,
  dataset,
  outgroup = NULL,
  outdataset = NULL,
  sd = NULL,
  mean = NULL,
  byrows = NULL,
  onmemory = NULL,
  wsize = NULL,
  overwrite = FALSE
)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing the dataset.
`dataset`	Character string. Name of the dataset to analyze.
`outgroup`	Character string, custom output group name (default: mean_sd)
`outdataset`	Character string, custom correlation dataset name (default: mean.dataset_original_name and sd.dataset_original_name)
`sd`	Logical (optional). Whether to compute sd. Default is TRUE.
`mean`	Logical (optional). Whether to compute mean. Default is TRUE.
`byrows`	Logical (optional). Whether to compute by rows (TRUE) or columns (FALSE). Default is FALSE.
`onmemory`	logical (default = FALSE). If TRUE, results are kept in memory and returned as a matrix; nothing is written to disk. If FALSE, results are written to disk.
`wsize`	Integer (optional). Block size for processing. Default is 1000.
`overwrite`	Logical (optional). Whether to overwrite existing results. Default is FALSE.

Details

This function provides efficient statistical computation capabilities with:

- Computation options:
  - Standard deviation computation
  - Mean computation
  - Row-wise or column-wise processing
- Processing features:
  - Block-based computation
  - Memory-efficient processing
  - Configurable block size
- Implementation features:
  - Safe HDF5 file operations
  - Memory-efficient implementation
  - Comprehensive error handling

Results are stored in a new group 'mean_sd' within the HDF5 file.

Value

Depending on the `onmemory` parameter:

If onmemory = TRUE List with components:

- `mean`: Numeric vector with column/row means (or NULL if not computed)
- `sd`: Numeric vector with column/row standard deviations (or NULL if not computed)

If onmemory = FALSE List with components:

- `fn`: Character string with the HDF5 filename
- `mean`: Character string with the full dataset path to the means (group/dataset)
- `sd`: Character string with the full dataset path to the standard deviations (group/dataset)

References

- The HDF Group. (2000-2010). HDF5 User's Guide.
- Welford, B. P. (1962). Note on a method for calculating corrected sums of squares and products. Technometrics, 4(3), 419-420.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
set.seed(123)
Y <- matrix(rnorm(100), 10, 10)
X <- matrix(rnorm(10), 10, 1)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", Y, "data", "matrix1",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix("test.hdf5", X, "data", "vector1",
                     overwriteFile = FALSE)

# Compute statistics
bdgetSDandMean_hdf5(
  filename = "test.hdf5",
  group = "data",
  dataset = "matrix1",
  sd = TRUE,
  mean = TRUE,
  byrows = TRUE,
  wsize = 500
)

# Cleanup
if (file.exists("test.hdf5")) {
  file.remove("test.hdf5")
}

## End(Not run)
```

`bdImportData_hdf5` *Import data from URL or file to HDF5 format*

Description

This function downloads data from a URL (if URL is provided) and decompresses it if needed, then imports the data into an HDF5 file. It supports both local files and remote URLs as input sources.

Usage

```
bdImportData_hdf5(
  inFile,
  destFile,
  destGroup,
  destDataset,
  header = TRUE,
  rownames = FALSE,
  overwrite = FALSE,
  overwriteFile = FALSE,
  sep = NULL,
  paral = NULL,
  threads = NULL
)
```

Arguments

`inFile`	Character string specifying either a local file path or URL containing the data to import
`destFile`	Character string specifying the file name and path where the HDF5 file will be stored
`destGroup`	Character string specifying the group name within the HDF5 file where the dataset will be stored
`destDataset`	Character string specifying the name for the dataset within the HDF5 file
`header`	Logical or character vector. If TRUE, the first row contains column names. If a character vector, use these as column names. Default is TRUE.
`rownames`	Logical or character vector. If TRUE, first column contains row names. If a character vector, use these as row names. Default is FALSE.
`overwrite`	Logical indicating if existing datasets should be overwritten. Default is FALSE.
`overwriteFile`	Logical indicating if the entire HDF5 file should be overwritten if it exists. CAUTION: This will delete all existing data. Default is FALSE.
`sep`	Character string specifying the field separator in the input file. Default is "\t" (tab).
`paral`	Logical indicating whether to use parallel computation. Default is TRUE.
`threads`	Integer specifying the number of threads to use for parallel computation. Only used if paral=TRUE. If NULL, uses maximum available threads.

Value

No return value. The function writes the data directly to the specified HDF5 file.

Examples

```
## Not run:
# Import from local file
bdImportData_hdf5(
  inFile = "data.txt",
  destFile = "output.h5",
  destGroup = "mydata",
  destDataset = "matrix1",
  header = TRUE,
  sep = "\t"
)
```

```
# Import from URL
bdImportData_hdf5(
  inFile = "https://example.com/data.csv",
  destFile = "output.h5",
  destGroup = "downloaded",
  destDataset = "remote_data",
  sep = ","
)

## End(Not run)
```

---

bdImportTextFile_hdf5 *Import Text File to HDF5*

Description

Converts a text file (e.g., CSV, TSV) to HDF5 format, providing efficient storage and access capabilities.

Usage

```
bdImportTextFile_hdf5(
  filename,
  outputfile,
  outGroup,
  outDataset,
  sep = NULL,
  header = FALSE,
  rownames = FALSE,
  overwrite = FALSE,
  paral = NULL,
  threads = NULL,
  overwriteFile = NULL
)
```

Arguments

filename	Character string. Path to the input text file.
outputfile	Character string. Path to the output HDF5 file.
outGroup	Character string. Name of the group to create in HDF5 file.
outDataset	Character string. Name of the dataset to create.
sep	Character string (optional). Field separator, default is "\t".
header	Logical (optional). Whether first row contains column names.
rownames	Logical (optional). Whether first column contains row names.

`overwrite`	Logical (optional). Whether to overwrite existing dataset.
`paral`	Logical (optional). Whether to use parallel processing.
`threads`	Integer (optional). Number of threads for parallel processing.
`overwriteFile`	Logical (optional). Whether to overwrite existing HDF5 file.

Details

This function provides flexible text file import capabilities with support for:

- Input format options:
  - Custom field separators
  - Header row handling
  - Row names handling
- Processing options:
  - Parallel processing
  - Memory-efficient import
  - Configurable thread count
- File handling:
  - Safe file operations
  - Overwrite protection
  - Comprehensive error handling

The function supports parallel processing for large files and provides memory-efficient import capabilities.

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the imported data (group/dataset)

ds_rows Character string with the full dataset path to the row names

ds_cols Character string with the full dataset path to the column names

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices directly

Examples

```
## Not run:
library(BigDataStatMeth)

# Create a test CSV file
data <- matrix(rnorm(100), 10, 10)
write.csv(data, "test.csv", row.names = FALSE)

# Import to HDF5
bdImportTextFile_hdf5(
  filename = "test.csv",
  outputfile = "output.hdf5",
  outGroup = "data",
  outDataset = "matrix1",
  sep = ",",
  header = TRUE,
  overwriteFile = TRUE
)

# Cleanup
unlink(c("test.csv", "output.hdf5"))

## End(Not run)
```

`bdImputeSNPs_hdf5` *Impute Missing SNP Values in HDF5 Dataset*

Description

Performs imputation of missing values in SNP (Single Nucleotide Polymorphism) data stored in HDF5 format.

Usage

```
bdImputeSNPs_hdf5(
  filename,
  group,
  dataset,
  outgroup = NULL,
  outdataset = NULL,
  bycols = TRUE,
  paral = NULL,
  threads = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing input dataset.
`dataset`	Character string. Name of the dataset to impute.
`outgroup`	Character string (optional). Output group path. If NULL, uses input group.
`outdataset`	Character string (optional). Output dataset name. If NULL, overwrites input dataset.
`bycols`	Logical (optional). Whether to impute by columns (TRUE) or rows (FALSE). Default is TRUE.
`paral`	Logical (optional). Whether to use parallel processing.
`threads`	Integer (optional). Number of threads for parallel processing.
`overwrite`	Logical (optional). Whether to overwrite existing dataset.

Details

This function provides efficient imputation capabilities for genomic data with support for:

- Imputation options:
  - Row-wise or column-wise imputation
  - Parallel processing
  - Configurable thread count
- Output options:
  - Custom output location
  - In-place modification
  - Overwrite protection
- Implementation features:
  - Memory-efficient processing
  - Safe file operations
  - Error handling

The function supports both in-place modification and creation of new datasets.

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the imputed data (group/dataset)

References

- The HDF Group. (2000-2010). HDF5 User's Guide.
- Li, Y., et al. (2009). Genotype Imputation. Annual Review of Genomics and Human Genetics, 10, 387-406.

See Also

- bdCreate_hdf5_matrix for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test data with missing values
data <- matrix(sample(c(0, 1, 2, NA), 100, replace = TRUE), 10, 10)

# Save to HDF5
fn <- "snp_data.hdf5"
bdCreate_hdf5_matrix(fn, data, "genotype", "snps",
                     overwriteFile = TRUE)

# Impute missing values
bdImputeSNPs_hdf5(
  filename = fn,
  group = "genotype",
  dataset = "snps",
  outgroup = "genotype_imputed",
  outdataset = "snps_complete",
  bycols = TRUE,
  paral = TRUE
)

# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

---

bdInvCholesky_hdf5 *Matrix Inversion using Cholesky Decomposition for HDF5-Stored Matrices*

Description

Computes the inverse of a symmetric positive-definite matrix stored in an HDF5 file using the Cholesky decomposition method. This approach is more efficient and numerically stable than general matrix inversion methods for symmetric positive-definite matrices.

Usage

```
bdInvCholesky_hdf5(
  filename,
  group,
  dataset,
  outdataset,
  outgroup = NULL,
  fullMatrix = NULL,
  overwrite = NULL,
  threads = 2L,
  elementsBlock = 1000000L
)
```

Arguments

`filename`	Character string. Path to the HDF5 file containing the input matrix.
`group`	Character string. Path to the group containing the input dataset.
`dataset`	Character string. Name of the input dataset to invert.
`outdataset`	Character string. Name for the output dataset.
`outgroup`	Character string. Optional output group path. If not provided, results are stored in the input group.
`fullMatrix`	Logical. If TRUE, stores the complete inverse matrix. If FALSE (default), stores only the lower triangular part to save space.
`overwrite`	Logical. If TRUE, allows overwriting existing results.
`threads`	Integer. Number of threads for parallel computation (default = 2).
`elementsBlock`	Integer. Maximum number of elements to process in each block (default = 1,000,000). For matrices larger than 5000x5000, automatically adjusted to number of rows or columns * 2.

Details

This function implements an efficient matrix inversion algorithm that leverages the special properties of symmetric positive-definite matrices. Key features:

- Uses Cholesky decomposition for improved numerical stability
- Block-based computation for large matrices
- Optional storage formats (full or triangular)
- Parallel processing support
- Memory-efficient block algorithm

The algorithm proceeds in two main steps:

1. Compute the Cholesky decomposition A = LL'
2. Solve the system LL'X = I for X = A^(-1)

Advantages of this method:

- More efficient than general matrix inversion
- Better numerical stability
- Preserves matrix symmetry
- Exploits positive-definiteness for efficiency

Value

List with components:

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the inverse Cholesky decomposition A^(-1) result (group/dataset)

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- Higham, N. J. (2002). Accuracy and Stability of Numerical Algorithms, 2nd Edition. SIAM.

See Also

- `bdCholesky_hdf5` for the underlying Cholesky decomposition
- `bdSolve_hdf5` for solving linear systems

Examples

```
## Not run:
library(rhdf5)

# Create a symmetric positive-definite matrix
set.seed(1234)
X <- matrix(rnorm(100), 10, 10)
A <- crossprod(X)  # A = X'X is symmetric positive-definite

# Save to HDF5
h5createFile("matrix.h5")
h5write(A, "matrix.h5", "data/matrix")

# Compute inverse using Cholesky decomposition
bdInvCholesky_hdf5("matrix.h5", "data", "matrix",
                   outdataset = "inverse",
                   outgroup = "results",
                   fullMatrix = TRUE,
                   threads = 4)

# Verify the inverse
Ainv <- h5read("matrix.h5", "results/inverse")
max(abs(A %*% Ainv - diag(nrow(A))))  # Should be very small

## End(Not run)
```

`bdIsLocked_hdf5` *Test whether an HDF5 file is locked (in use)*

Description

Uses HDF5 file locking to check if `filename` can be opened in read/write mode. If opening fails under locking, the file is treated as "in use" and `TRUE` is returned. Non-existent files return `FALSE`.

Usage

```
bdIsLocked_hdf5(filename)
```

Arguments

`filename`	Character. Path to the HDF5 file.

Details

Requires HDF5 file locking (HDF5 >= 1.12 recommended). The function sets `HDF5_USE_FILE_LOCKING=TRUE` for the process.

Value

Logical scalar: `TRUE` if locked/in use, `FALSE` otherwise.

Examples

```
## Not run:
if (bdIsFileLocked("data.h5")) stop("File in use")

## End(Not run)
```

`bdmove_hdf5_dataset` *Move HDF5 Dataset*

Description

Moves an HDF5 dataset from one location to another within the same HDF5 file. This function automatically handles moving associated rownames and colnames datasets, creates parent groups if needed, and updates all internal references.

Usage

```
bdmove_hdf5_dataset(filename, source_path, dest_path, overwrite = FALSE)
```

Arguments

`filename`	Character string. Path to the HDF5 file
`source_path`	Character string. Current path to the dataset (e.g., "/group1/dataset1")
`dest_path`	Character string. New path for the dataset (e.g., "/group2/new_name")
`overwrite`	Logical. Whether to overwrite destination if it exists (default: FALSE)

Details

This function provides a high-level interface for moving datasets within HDF5 files. The operation is efficient as it uses HDF5's native linking mechanism without copying actual data.

Key features:

- Moves main dataset and associated rownames/colnames datasets
- Creates parent directory structure automatically
- Preserves all dataset attributes and properties
- Updates internal dataset references
- Efficient metadata-only operation
- Comprehensive error handling

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the moved dataset in its new location (group/dataset)

Behavior

- If the destination parent groups don't exist, they will be created automatically
- Associated rownames and colnames datasets are moved to the same new group
- All dataset attributes and properties are preserved during the move
- The operation is atomic - either all elements move successfully or none do

Requirements

- The HDF5 file must exist and be accessible
- The source dataset must exist
- The file must not be locked by another process
- User must have read-write permissions on the file

Author(s)

BigDataStatMeth package authors

See Also

Other BigDataStatMeth HDF5 utilities: `bdsubset_hdf5_dataset()`

Examples

```
## Not run:
# Move dataset to a different group
success <- bdmove_hdf5_dataset("data.h5",
                          source_path = "/old_group/my_dataset",
                          dest_path = "/new_group/my_dataset")

# Rename dataset within the same group
success <- bdmove_hdf5_dataset("data.h5",
                          source_path = "/data/old_name",
                          dest_path = "/data/new_name",
                          overwrite = TRUE)

# Move dataset to root level
success <- bdmove_hdf5_dataset("data.h5",
                          source_path = "/deep/nested/dataset",
                          dest_path = "/dataset")

# Move with automatic group creation
success <- bdmove_hdf5_dataset("data.h5",
                          source_path = "/old_location/dataset",
                          dest_path = "/new/deep/structure/dataset")

## End(Not run)
```

`bdNormalize_hdf5` *Normalize dataset in HDF5 file*

Description

Performs block-wise normalization of datasets stored in HDF5 format through centering and/or scaling operations. Supports both row-wise and column-wise normalization with memory-efficient block processing.

Usage

```
bdNormalize_hdf5(
  filename,
  group,
  dataset,
  bcenter = NULL,
  bscale = NULL,
  byrows = NULL,
  wsize = NULL,
  overwrite = FALSE
)
```

Arguments

`filename`	String indicating the HDF5 file path
`group`	String specifying the group containing the dataset
`dataset`	String specifying the dataset name to normalize
`bcenter`	Optional boolean indicating whether to center the data. If TRUE (default), subtracts mean from each column/row
`bscale`	Optional boolean indicating whether to scale the data. If TRUE (default), divides by standard deviation
`byrows`	Optional boolean indicating whether to operate by rows. If TRUE, processes row-wise; if FALSE (default), column-wise
`wsize`	Optional integer specifying the block size for processing. Default is 1000
`overwrite`	Optional boolean indicating whether to overwrite existing datasets. Default is false

Details

The function implements block-wise normalization through:

Statistical computations:

- Mean calculation (for centering)
- Standard deviation calculation (for scaling)
- Efficient block-wise updates

Memory efficiency:

- Block-wise data processing
- Minimal temporary storage
- Proper resource cleanup

Processing options:

- Row-wise or column-wise operations
- Flexible block size selection
- Optional centering and scaling

Error handling:

- Input validation
- Resource management
- Exception handling

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string. Path to the HDF5 file containing the results

ds Character string. Full dataset path to the normalized data, stored under "NORMALIZED/\[group\]/\[dataset\]"

mean Character string. Dataset path to the column means used for centering, stored under "NORMALIZED/\[group\]/mean.\[dataset\]"

sd Character string. Dataset path to the standard deviations used for scaling, stored under "NORMALIZED/\[group\]/sd.\[dataset\]"

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test data
data <- matrix(rnorm(1000*100), 1000, 100)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", data, "data", "matrix",
                     overwriteFile = TRUE)

# Normalize data
bdNormalize_hdf5("test.hdf5", "data", "matrix",
                 bcenter = TRUE,
                 bscale = TRUE,
                 wsize = 1000)

## End(Not run)
```

`bdPCA_hdf5` *Principal Component Analysis for HDF5-Stored Matrices*

Description

Performs Principal Component Analysis (PCA) on a large matrix stored in an HDF5 file. PCA reduces the dimensionality of the data while preserving as much variance as possible. The implementation uses SVD internally for efficient and numerically stable computation.

Usage

```
bdPCA_hdf5(
  filename,
  group,
  dataset,
  ncomponents = 0L,
  bcenter = FALSE,
  bscale = FALSE,
  k = 2L,
  q = 1L,
  rankthreshold = 0,
  SVDgroup = NULL,
  overwrite = FALSE,
  method = NULL,
  threads = NULL
)
```

Arguments

`filename`	Character string. Path to the HDF5 file containing the input matrix.
`group`	Character string. Path to the group containing the input dataset.
`dataset`	Character string. Name of the input dataset to analyze.
`ncomponents`	Integer. Number of principal components to compute (default = 0, which computes all components).
`bcenter`	Logical. If TRUE, centers the data by subtracting column means. Default is FALSE.
`bscale`	Logical. If TRUE, scales the centered columns by their standard deviations (if centered) or root mean square. Default is FALSE.
`k`	Integer. Number of local SVDs to concatenate at each level (default = 2). Controls memory usage in block computation.
`q`	Integer. Number of levels for SVD computation (default = 1). Higher values can improve accuracy but increase computation time.
`rankthreshold`	Numeric. Threshold for determining matrix rank (default = 0). Must be between 0 and 0.1.
`SVDgroup`	Character string. Group name where intermediate SVD results are stored. If SVD was previously computed, results will be reused from this group.
`overwrite`	Logical. If TRUE, forces recomputation of SVD even if results exist.
`method`	Character string. Computation method: • "auto": Automatically selects method based on matrix size • "blocks": Uses block-based computation (for large matrices) • "full": Performs direct computation (for smaller matrices)
`threads`	Integer. Number of threads for parallel computation.

Details

This function implements a scalable PCA algorithm suitable for large matrices that may not fit in memory. Key features include:

- Automatic method selection based on matrix size
- Block-based computation for large matrices

- Optional data preprocessing (centering and scaling)
- Parallel processing support
- Memory-efficient incremental algorithm
- Reuse of existing SVD results

The implementation uses SVD internally and supports two computation methods:

- Full decomposition: Suitable for matrices that fit in memory
- Block-based decomposition: For large matrices, uses an incremental algorithm

Value

A list containing the paths to the PCA results stored in the HDF5 file:

fn Character string. Path to the HDF5 file containing the results

lambda Character string. Dataset path to eigenvalues λ

variance Character string. Dataset path to variance explained by each PC

cumvar Character string. Dataset path to cumulative variance explained

var.coord Character string. Dataset path to variable coordinates on the PCs

var.cos2 Character string. Dataset path to squared cosines (quality of representation) for variables

ind.dist Character string. Dataset path to distances of individuals from the origin

components Character string. Dataset path to principal components (rotated data)

ind.coord Character string. Dataset path to individual coordinates on the PCs

ind.cos2 Character string. Dataset path to squared cosines (quality of representation) for individuals

ind.contrib Character string. Dataset path to contributions of individuals to each PC

All results are written to the HDF5 file in the group 'PCA/`dataset`'.

References

- Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review, 53(2), 217-288.
- Jolliffe, I. T. (2002). Principal Component Analysis, Second Edition. Springer Series in Statistics.

See Also

- `bdSVD_hdf5` for the underlying SVD computation
- `bdNormalize_hdf5` for data preprocessing options

Examples

```
## Not run:
# Create a sample large matrix in HDF5
library(rhdf5)
X <- matrix(rnorm(10000), 1000, 10)
h5createFile("data.h5")
h5write(X, "data.h5", "data/matrix")

# Basic PCA with default parameters
bdPCA_hdf5("data.h5", "data", "matrix")

# PCA with preprocessing and specific number of components
bdPCA_hdf5("data.h5", "data", "matrix",
           ncomponents = 3,
           bcenter = TRUE, bscale = TRUE,
           method = "blocks",
           threads = 4)

## End(Not run)
```

`bdpseudoinv` *Compute Matrix Pseudoinverse (In-Memory)*

Description

Computes the Moore-Penrose pseudoinverse of a matrix using SVD decomposition. This implementation handles both square and rectangular matrices, and provides numerically stable results even for singular or near-singular matrices.

Usage

```
bdpseudoinv(X, threads = NULL)
```

Arguments

`X`	Numeric matrix or vector to be pseudoinverted.
`threads`	Optional integer. Number of threads for parallel computation. If NULL, uses maximum available threads.

Details

The Moore-Penrose pseudoinverse (denoted A+) of a matrix A is computed using Singular Value Decomposition (SVD).

For a matrix A = U*Sigma*V^T (where ^T denotes transpose), the pseudoinverse is computed as:

$$A^+ = V\Sigma^+U^T$$

where Sigma+ is obtained by taking the reciprocal of non-zero singular values.

Value

The pseudoinverse matrix of X.

Mathematical Details

- SVD decomposition: $A = U\Sigma V^T$
- Pseudoinverse: $A^+ = V\Sigma^+U^T$
- $\Sigma^+_{ii} = 1/\Sigma_{ii}$ if $\Sigma_{ii} >$ tolerance
- $\Sigma^+_{ii} = 0$ otherwise

Key features:

- Robust computation:
  - Handles singular and near-singular matrices
  - Automatic threshold for small singular values
  - Numerically stable implementation
- Implementation details:
  - Uses efficient SVD algorithms
  - Parallel processing support
  - Memory-efficient computation
  - Handles both dense and sparse inputs

The pseudoinverse satisfies the Moore-Penrose conditions:

- $AA^+A = A$
- $A^+AA^+ = A^+$
- $(AA^+)^* = AA^+$
- $(A^+A)^* = A^+A$

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- Ben-Israel, A., & Greville, T. N. E. (2003). Generalized Inverses: Theory and Applications, 2nd Edition. Springer.

See Also

- `bdpseudoinv_hdf5` for HDF5-stored matrices
- `bdSVD_hdf5` for singular value decomposition

Examples

```
library(BigDataStatMeth)

# Create a singular matrix
X <- matrix(c(1,2,3,2,4,6), 2, 3)  # rank-deficient matrix

# Compute pseudoinverse
X_pinv <- bdpseudoinv(X)

# Verify Moore-Penrose conditions
# 1. X %*% X_pinv %*% X = X
all.equal(X %*% X_pinv %*% X, X)

# 2. X_pinv %*% X %*% X_pinv = X_pinv
all.equal(X_pinv %*% X %*% X_pinv, X_pinv)
```

`bdpseudoinv_hdf5` *Compute Matrix Pseudoinverse (HDF5-Stored)*

Description

Computes the Moore-Penrose pseudoinverse of a matrix stored in HDF5 format. The implementation is designed for large matrices, using block-based processing and efficient I/O operations.

Usage

```
bdpseudoinv_hdf5(
  filename,
  group,
  dataset,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = NULL,
  threads = NULL
)
```

Arguments

`filename`	String. Path to the HDF5 file.
`group`	String. Group containing the input matrix.
`dataset`	String. Dataset name for the input matrix.
`outgroup`	Optional string. Output group name (defaults to "PseudoInverse").
`outdataset`	Optional string. Output dataset name (defaults to input dataset name).
`overwrite`	Logical. Whether to overwrite existing results.
`threads`	Optional integer. Number of threads for parallel computation.

Details

This function provides an HDF5-based implementation for computing pseudoinverses of large matrices. Key features:

- HDF5 Integration:
  - Efficient reading of input matrix
  - Block-based processing for large matrices
  - Memory-efficient computation
  - Direct output to HDF5 format
- Implementation Features:
  - SVD-based computation
  - Parallel processing support
  - Automatic memory management
  - Flexible output options

The function handles:

- Data validation
- Memory management
- Error handling
- HDF5 file operations

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the pseudoinverse matrix (group/dataset)

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdpseudoinv` for in-memory computation
- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
library(BigDataStatMeth)

# Create a singular matrix
X <- matrix(c(1,2,3,2,4,6), 2, 3)
fn <- "test.hdf5"

# Save to HDF5
bdCreate_hdf5_matrix(filename = fn,
                     object = X,
                     group = "data",
                     dataset = "X",
                     overwriteFile = TRUE)

# Compute pseudoinverse
bdpseudoinv_hdf5(filename = fn,
                 group = "data",
                 dataset = "X",
                 outgroup = "results",
                 outdataset = "X_pinv",
                 overwrite = TRUE)

# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}
```

`bdQR` *QR Decomposition for In-Memory Matrices*

Description

Computes the QR decomposition (also called QR factorization) of a matrix A into a product A = QR where Q is an orthogonal matrix and R is an upper triangular matrix. This function operates on in-memory matrices.

Usage

```
bdQR(X, thin = NULL, block_size = NULL, threads = NULL)
```

Arguments

`X`	A real matrix or vector to be decomposed
`thin`	Logical. If TRUE, returns the reduced (thin) Q matrix. If FALSE (default), returns the full Q matrix. The thin decomposition is more memory efficient.
`block_size`	Integer. Optional block size for blocked computation. Larger blocks may improve performance but require more memory.
`threads`	Integer. Optional number of threads for parallel computation. If NULL, uses all available threads.

Details

The QR decomposition is a fundamental matrix factorization that decomposes a matrix into an orthogonal matrix Q and an upper triangular matrix R. This implementation:

- Supports both thin and full QR decomposition
- Can utilize parallel computation for better performance
- Handles both matrix and vector inputs

Value

A list containing:

- Q: The orthogonal matrix Q
- R: The upper triangular matrix R

See Also

`bdQR_hdf5` for QR decomposition of HDF5-stored matrices

Examples

```
## Not run:
# Create a random 100x50 matrix
X <- matrix(rnorm(5000), 100, 50)

# Compute thin QR decomposition
result <- bdQR(X, thin = TRUE)

# Verify the decomposition
# Should be approximately zero
max(abs(X - result$Q %*% result$R))

## End(Not run)
```

`bdQR_hdf5` *QR Decomposition for HDF5-Stored Matrices*

Description

Computes the QR decomposition of a matrix stored in an HDF5 file, factoring it into a product A = QR where Q is an orthogonal matrix and R is an upper triangular matrix. Results are stored back in the HDF5 file.

Usage

```
bdQR_hdf5(
  filename,
  group,
  dataset,
  outgroup = NULL,
  outdataset = NULL,
  thin = NULL,
  block_size = NULL,
  overwrite = NULL,
  threads = NULL
)
```

Arguments

`filename`	Character string. Path to the HDF5 file containing the input matrix.
`group`	Character string. Path to the group containing the input dataset.
`dataset`	Character string. Name of the input dataset to decompose.
`outgroup`	Character string. Optional output group path where results will be stored. If not provided, results are stored in `<input_group>/QRDec`.
`outdataset`	Character string. Optional base name for output datasets. Results will be stored as `Q.'outdataset'` and `R.'outdataset'`.
`thin`	Logical. If TRUE, computes the reduced (thin) QR decomposition. If FALSE (default), computes the full decomposition.
`block_size`	Integer. Optional block size for blocked computation.
`overwrite`	Logical. If TRUE, allows overwriting existing datasets. Default is FALSE.
`threads`	Integer. Optional number of threads for parallel computation. If NULL, uses all available threads.

Details

This function performs QR decomposition on large matrices stored in HDF5 format, which is particularly useful for matrices too large to fit in memory. Features include:

- Support for both thin and full QR decomposition
- Blocked computation for improved performance
- Parallel processing capabilities
- Flexible output location specification
- Optional overwriting of existing datasets

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds_Q Character string with the full dataset path to the Q matrix (orthogonal matrix). Results are written to the HDF5 file as "Q.'outdataset'" within the specified group

ds_R Character string with the full dataset path to the R matrix (upper triangular matrix). Results are written to the HDF5 file as "R.'outdataset'" within the specified group

See Also

`bdQR` for QR decomposition of in-memory matrices

Examples

```
## Not run:
# Create a sample HDF5 file with a matrix
library(rhdf5)
A <- matrix(rnorm(1000), 100, 10)
h5createFile("example.h5")
h5write(A, "example.h5", "mygroup/mymatrix")

# Compute QR decomposition
bdQR_hdf5("example.h5", "mygroup", "mymatrix",
          outgroup = "mygroup/results",
          outdataset = "qr_result",
          thin = TRUE)

## End(Not run)
```

`bdReduce_hdf5_dataset` *Reduce Multiple HDF5 Datasets*

Description

Reduces multiple datasets within an HDF5 group using arithmetic operations (addition or subtraction).

Usage

```
bdReduce_hdf5_dataset(
  filename,
  group,
  reducefunction,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = FALSE,
  remove = FALSE
)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing datasets.
`reducefunction`	Character. Operation to apply, either "+" or "-".
`outgroup`	Character string (optional). Output group path. If NULL, uses input group.
`outdataset`	Character string (optional). Output dataset name. If NULL, uses input group name.
`overwrite`	Logical (optional). Whether to overwrite existing dataset. Default is FALSE.
`remove`	Logical (optional). Whether to remove source datasets after reduction. Default is FALSE.

Details

This function provides efficient dataset reduction capabilities with:

- Operation options:
  - Addition of datasets
  - Subtraction of datasets
- Output options:
  - Custom output location
  - Configurable dataset name
  - Overwrite protection
- Implementation features:
  - Memory-efficient processing
  - Safe file operations
  - Optional source cleanup
  - Comprehensive error handling

The function processes datasets efficiently while maintaining data integrity.

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the reduced dataset (group/dataset)

func Character string with the reduction function applied

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
X1 <- matrix(1:100, 10, 10)
X2 <- matrix(101:200, 10, 10)
X3 <- matrix(201:300, 10, 10)

# Save to HDF5
fn <- "test.hdf5"
bdCreate_hdf5_matrix(fn, X1, "data", "matrix1",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix(fn, X2, "data", "matrix2",
                     overwriteFile = FALSE)
bdCreate_hdf5_matrix(fn, X3, "data", "matrix3",
                     overwriteFile = FALSE)

# Reduce datasets by addition
bdReduce_hdf5_dataset(
  filename = fn,
  group = "data",
  reducefunction = "+",
  outgroup = "results",
  outdataset = "sum_matrix",
  overwrite = TRUE
)

# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

---

bdRemovelowdata_hdf5 *Remove Low-Representation SNPs from HDF5 Dataset*

Description

Removes SNPs (Single Nucleotide Polymorphisms) with low representation from genomic data stored in HDF5 format.

Usage

```
bdRemovelowdata_hdf5(
  filename,
  group,
  dataset,
  outgroup,
  outdataset,
  pcent,
  bycols,
  overwrite = NULL
)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing input dataset.
`dataset`	Character string. Name of the dataset to filter.
`outgroup`	Character string. Output group path for filtered data.
`outdataset`	Character string. Output dataset name for filtered data.
`pcent`	Numeric (optional). Threshold percentage for removal (0-1). Default is 0.5. SNPs with representation below this threshold are removed.
`bycols`	Logical (optional). Whether to filter by columns (TRUE) or rows (FALSE). Default is TRUE.
`overwrite`	Logical (optional). Whether to overwrite existing dataset. Default is FALSE.

Details

This function provides efficient filtering capabilities for genomic data with support for:

- Filtering options:
  - Row-wise or column-wise filtering
  - Configurable threshold percentage
  - Flexible output location
- Implementation features:
  - Memory-efficient processing
  - Safe file operations
  - Comprehensive error handling
  - Progress reporting

The function supports both in-place modification and creation of new datasets.

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the filtered dataset (group/dataset)

nremoved Integer with the number of rows/columns removed due to low data quality

References

- The HDF Group. (2000-2010). HDF5 User's Guide.
- Marchini, J., & Howie, B. (2010). Genotype imputation for genome-wide association studies. Nature Reviews Genetics, 11(7), 499-511.

See Also

- `bdImputeSNPs_hdf5` for imputing missing SNP values
- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test SNP data with missing values
snps <- matrix(sample(c(0, 1, 2, NA), 100, replace = TRUE,
                      prob = c(0.3, 0.3, 0.3, 0.1)), 10, 10)

# Save to HDF5
fn <- "snp_data.hdf5"
bdCreate_hdf5_matrix(fn, snps, "genotype", "raw_snps",
                      overwriteFile = TRUE)

# Remove SNPs with low representation
bdRemovelowdata_hdf5(
  filename = fn,
  group = "genotype",
  dataset = "raw_snps",
  outgroup = "genotype_filtered",
  outdataset = "filtered_snps",
  pcent = 0.3,
  bycols = TRUE
)

# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

`bdRemoveMAF_hdf5` *Remove SNPs Based on Minor Allele Frequency*

Description

Filters SNPs (Single Nucleotide Polymorphisms) based on Minor Allele Frequency (MAF) in genomic data stored in HDF5 format.

Usage

```
bdRemoveMAF_hdf5(
  filename,
  group,
  dataset,
  outgroup,
  outdataset,
  maf,
  bycols,
  blocksize,
  overwrite = NULL
)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing input dataset.
`dataset`	Character string. Name of the dataset to filter.
`outgroup`	Character string. Output group path for filtered data.
`outdataset`	Character string. Output dataset name for filtered data.
`maf`	Numeric (optional). MAF threshold for filtering (0-1). Default is 0.05. SNPs with MAF above this threshold are removed.
`bycols`	Logical (optional). Whether to process by columns (TRUE) or rows (FALSE). Default is FALSE.
`blocksize`	Integer (optional). Block size for processing. Default is 100. Larger values use more memory but may be faster.
`overwrite`	Logical (optional). Whether to overwrite existing dataset. Default is FALSE.

Details

This function provides efficient MAF-based filtering capabilities with:

- Filtering options:
  - MAF threshold-based filtering
  - Row-wise or column-wise processing
  - Block-based processing
- Implementation features:
  - Memory-efficient processing
  - Block-based operations
  - Safe file operations
  - Progress reporting

The function supports both in-place modification and creation of new datasets.

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the filtered dataset (group/dataset)

nremoved Integer with the number of SNPs removed due to low Minor Allele Frequency (MAF)

References

- The HDF Group. (2000-2010). HDF5 User's Guide.
- Marees, A. T., et al. (2018). A tutorial on conducting genome-wide association studies: Quality control and statistical analysis. International Journal of Methods in Psychiatric Research, 27(2), e1608.

See Also

- `bdRemovelowdata_hdf5` for removing low-representation SNPs
- `bdImputeSNPs_hdf5` for imputing missing SNP values

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test SNP data
snps <- matrix(sample(c(0, 1, 2), 1000, replace = TRUE,
                     prob = c(0.7, 0.2, 0.1)), 100, 10)

# Save to HDF5
fn <- "snp_data.hdf5"
bdCreate_hdf5_matrix(fn, snps, "genotype", "raw_snps",
                     overwriteFile = TRUE)

# Remove SNPs with high MAF
bdRemoveMAF_hdf5(
  filename = fn,
  group = "genotype",
  dataset = "raw_snps",
  outgroup = "genotype_filtered",
  outdataset = "filtered_snps",
  maf = 0.1,
  bycols = TRUE,
  blocksize = 50
)

# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}
```

```
## End(Not run)
```

bdRemove_hdf5_element *Remove Elements from HDF5 File*

Description

Removes specified groups or datasets from an HDF5 file.

Usage

```
bdRemove_hdf5_element(filename, elements)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`elements`	Character vector. Full paths to elements to remove (e.g., "group/dataset" or "group/subgroup").

Details

This function provides safe element removal capabilities with:

- Removal options:
  - Single element removal
  - Multiple element removal
  - Groups and datasets removal
- Implementation features:
  - Safe file operations
  - Memory-efficient implementation
  - Comprehensive error handling
  - Path validation

The function validates paths and performs safe removal operations.

Value

No return value, called for side effects (element removal).

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrices
matA <- matrix(1:15, nrow = 3, byrow = TRUE)
matB <- matrix(15:1, nrow = 3, byrow = TRUE)

# Save to HDF5
fn <- "test.hdf5"
bdCreate_hdf5_matrix(fn, matA, "data", "matrix1",
                     overwriteFile = TRUE)
bdCreate_hdf5_matrix(fn, matB, "data", "matrix2",
                     overwriteFile = FALSE)

# Remove elements
bdRemove_hdf5_element(fn, c("data/matrix1", "data/matrix2"))

# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

bdScalarwproduct *Matrix–scalar weighted product*

Description

Multiplies a numeric matrix A by a scalar weight w, returning $w * A$. The input must be a base R numeric matrix (or convertible to one).

Usage

```
bdScalarwproduct(A, w)
```

Arguments

A	Numeric matrix (or object convertible to a dense numeric matrix).
w	Numeric scalar weight.

Value

A numeric matrix with the same dimensions as A.

Examples

```
set.seed(1234)
n <- 5; p <- 3
X <- matrix(rnorm(n * p), n, p)
w <- 0.75
bdScalarwproduct(X, w)
```

`bdSolve` *Solve Linear System AX = B (In-Memory)*

Description

Solves the linear system AX = B where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The function automatically detects if A is symmetric and uses the appropriate solver.

Usage

```
bdSolve(A, B)
```

Arguments

`A`	Numeric matrix. The coefficient matrix (must be square).
`B`	Numeric matrix. The right-hand side matrix (must have same number of rows as A).

Details

This function provides an efficient implementation for solving linear systems using LAPACK routines. Key features:

- Automatic detection of matrix properties:
  - Checks for matrix symmetry
  - Selects optimal solver based on matrix structure
- Solver selection:
  - Symmetric systems: Uses LAPACK's dsysv routine
  - Non-symmetric systems: Uses LAPACK's dgesv routine
- Performance optimizations:
  - Automatic workspace sizing
  - Efficient memory management
  - Support for multiple right-hand sides

The implementation ensures:

- Robust error handling
- Efficient memory usage
- Numerical stability
- Support for various matrix sizes

Value

Numeric matrix X, the solution to AX = B.

References

- Anderson, E. et al. (1999). LAPACK Users' Guide, 3rd Edition. SIAM, Philadelphia.
- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.

See Also

- `bdSolve_hdf5` for solving systems with HDF5-stored matrices
- `solve` for R's built-in solver

Examples

```
library(BigDataStatMeth)

# Create test matrices
n <- 500
m <- 500

A <- matrix(runif(n*m), nrow = n, ncol = m)
B <- matrix(runif(n), nrow = n)
AS <- A %*% t(A)  # Create symmetric matrix

# Solve using bdSolve
X <- bdSolve(A, B)

# Compare with R's solve
XR <- solve(A, B)
all.equal(X, XR, check.attributes=FALSE)
```

`bdSolve_hdf5` *Solve Linear System AX = B (HDF5-Stored)*

Description

Solves the linear system AX = B where matrices A and B are stored in HDF5 format. The solution X is written back to the HDF5 file.

Usage

```
bdSolve_hdf5(
  filename,
  groupA,
  datasetA,
  groupB,
  datasetB,
  outgroup = NULL,
  outdataset = NULL,
  overwrite = NULL
)
```

Arguments

`filename`	String. Path to the HDF5 file.
`groupA`	String. Group containing matrix A.
`datasetA`	String. Dataset name for matrix A.
`groupB`	String. Group containing matrix B.
`datasetB`	String. Dataset name for matrix B.
`outgroup`	Optional string. Output group name (defaults to "Solved").
`outdataset`	Optional string. Output dataset name (defaults to "A_B").
`overwrite`	Logical. Whether to overwrite existing results.

Details

This function provides an HDF5-based implementation for solving large linear systems. Key features:

- HDF5 Integration:
  - Efficient reading of input matrices
  - Memory-efficient processing
  - Direct output to HDF5 format
- Implementation Features:
  - Automatic solver selection
  - Support for large matrices
  - Flexible output options
  - Memory-efficient processing

The function handles:

- Data validation
- Memory management
- Error handling
- HDF5 file operations

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the solution of the linear system (group/dataset)

References

- Anderson, E. et al. (1999). LAPACK Users' Guide, 3rd Edition. SIAM, Philadelphia.
- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdSolve` for in-memory matrix solving
- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
library(BigDataStatMeth)

# Create test matrices
N <- 1000
M <- 1000
fn <- "test_temp.hdf5"

set.seed(555)
Y <- matrix(rnorm(N*M), N, M)
X <- matrix(rnorm(N), N, 1)
Ycp <- crossprod(Y)

# Compare with in-memory solution
resm <- bdSolve(Ycp, X)
resr <- solve(Ycp, X)
all.equal(resm, resr)

# Save matrices to HDF5
bdCreate_hdf5_matrix(filename = fn,
                     object = Ycp,
                     group = "data",
                     dataset = "A",
                     transp = FALSE,
                     overwriteFile = TRUE,
                     overwriteDataset = TRUE,
                     unlimited = FALSE)

bdCreate_hdf5_matrix(filename = fn,
                     object = X,
                     group = "data",
                     dataset = "B",
                     transp = FALSE,
                     overwriteFile = FALSE,
                     overwriteDataset = TRUE,
                     unlimited = FALSE)

# Solve using HDF5-stored matrices
bdSolve_hdf5(filename = fn,
             groupA = "data",
             datasetA = "A",
              groupB = "data",
              datasetB = "B",
              outgroup = "Solved",
              outdataset = "A_B",
              overwrite = TRUE)

# Cleanup
if (file.exists(fn)) {
    file.remove(fn)
}
```

---

bdSort_hdf5_dataset *Sort HDF5 Dataset Using Predefined Order*

Description

Sorts a dataset in an HDF5 file based on a predefined ordering specified through a list of sorting blocks.

Usage

```
bdSort_hdf5_dataset(
  filename,
  group,
  dataset,
  outdataset,
  blockedSortlist,
  func,
  outgroup = NULL,
  overwrite = FALSE
)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing input dataset.
`dataset`	Character string. Name of the dataset to sort.
`outdataset`	Character string. Name for the sorted dataset.
`blockedSortlist`	List of data frames. Each data frame specifies the sorting order for a block of elements. See Details for structure.
`func`	Character string. Function to apply: • "sortRows" for row-wise sorting • "sortCols" for column-wise sorting
`outgroup`	Character string (optional). Output group path. If NULL, uses input group.
`overwrite`	Logical (optional). Whether to overwrite existing dataset. Default is FALSE.

Details

This function provides efficient dataset sorting capabilities with:

- Sorting options:
  - Row-wise sorting
  - Column-wise sorting
  - Block-based processing
- Implementation features:
  - Memory-efficient processing
  - Block-based operations
  - Safe file operations
  - Progress reporting

The sorting order is specified through a list of data frames, where each data frame represents a block of elements to be sorted. Each data frame must contain:

- Row names (current identifiers)
- chr (new identifiers)
- order (current positions)
- newOrder (target positions)

Example sorting blocks structure:

Block 1 (maintaining order): chr order newOrder Diagonal TCGA-OR-A5J1 TCGA-OR-A5J1 1 1 1 TCGA-OR-A5J2 TCGA-OR-A5J2 2 2 1 TCGA-OR-A5J3 TCGA-OR-A5J3 3 3 1 TCGA-OR-A5J4 TCGA-OR-A5J4 4 4 1

Block 2 (reordering with new identifiers): chr order newOrder Diagonal TCGA-OR-A5J5 TCGA-OR-A5JA 10 5 1 TCGA-OR-A5J6 TCGA-OR-A5JB 11 6 1 TCGA-OR-A5J7 TCGA-OR-A5JC 12 7 0 TCGA-OR-A5J8 TCGA-OR-A5JD 13 8 1

Block 3 (reordering with identifier swaps): chr order newOrder Diagonal TCGA-OR-A5J9 TCGA-OR-A5J5 5 9 1 TCGA-OR-A5JA TCGA-OR-A5J6 6 10 1 TCGA-OR-A5JB TCGA-OR-A5J7 7 11 1 TCGA-OR-A5JC TCGA-OR-A5J8 8 12 1 TCGA-OR-A5JD TCGA-OR-A5J9 9 13 0

In this example:

- Block 1 maintains the original order
- Block 2 assigns new identifiers (A5JA-D) to elements
- Block 3 swaps identifiers between elements
- The Diagonal column indicates whether the element is on the diagonal (1) or not (0)

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the sorted dataset (group/dataset)

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test data
data <- matrix(rnorm(100), 10, 10)
rownames(data) <- paste0("TCGA-OR-A5J", 1:10)

# Save to HDF5
fn <- "test.hdf5"
bdCreate_hdf5_matrix(fn, data, "data", "matrix1",
                     overwriteFile = TRUE)

# Create sorting blocks
block1 <- data.frame(
  chr = paste0("TCGA-OR-A5J", c(2,1,3,4)),
  order = 1:4,
  newOrder = c(2,1,3,4),
  row.names = paste0("TCGA-OR-A5J", 1:4)
)

block2 <- data.frame(
  chr = paste0("TCGA-OR-A5J", c(6,5,8,7)),
  order = 5:8,
  newOrder = c(6,5,8,7),
  row.names = paste0("TCGA-OR-A5J", 5:8)
)

# Sort dataset
bdSort_hdf5_dataset(
  filename = fn,
  group = "data",
  dataset = "matrix1",
  outdataset = "matrix1_sorted",
  blockedSortlist = list(block1, block2),
  func = "sortRows"
)

# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

`bdSplit_matrix_hdf5` *Split HDF5 Dataset into Submatrices*

Description

Splits a large dataset in an HDF5 file into smaller submatrices, with support for both row-wise and column-wise splitting.

Usage

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

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing input dataset.
`dataset`	Character string. Name of the dataset to split.
`outgroup`	Character string (optional). Output group path. If NULL, uses input group.
`outdataset`	Character string (optional). Base name for output datasets. If NULL, uses input dataset name with block number suffix.
`nblocks`	Integer (optional). Number of blocks to split into. Mutually exclusive with blocksize.
`blocksize`	Integer (optional). Size of each block. Mutually exclusive with nblocks.
`bycols`	Logical (optional). Whether to split by columns (TRUE) or rows (FALSE). Default is TRUE.
`overwrite`	Logical (optional). Whether to overwrite existing datasets. Default is FALSE.

Details

This function provides efficient dataset splitting capabilities with:

- Splitting options:
  - Row-wise or column-wise splitting

  - Fixed block size splitting
  - Fixed block count splitting
- Implementation features:
  - Memory-efficient processing
  - Block-based operations
  - Safe file operations
  - Progress reporting

The function supports two splitting strategies:

1. By number of blocks: Splits the dataset into a specified number of roughly equal-sized blocks
2. By block size: Splits the dataset into blocks of a specified size

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the output group path where the split datasets are stored. Multiple datasets are created in this location named as \<outdataset\>.1, \<outdataset\>.2, etc.

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test data
data <- matrix(rnorm(1000), 100, 10)

# Save to HDF5
fn <- "test.hdf5"
bdCreate_hdf5_matrix(fn, data, "data", "matrix1",
                     overwriteFile = TRUE)

# Split by number of blocks
bdSplit_matrix_hdf5(
  filename = fn,
  group = "data",
  dataset = "matrix1",
  outgroup = "data_split",
  outdataset = "block",
  nblocks = 4,
  bycols = TRUE
)

# Split by block size
bdSplit_matrix_hdf5(
  filename = fn,
  group = "data",
  dataset = "matrix1",
  outgroup = "data_split2",
  outdataset = "block",
  blocksize = 25,
  bycols = TRUE
)

# Cleanup
if (file.exists(fn)) {
  file.remove(fn)
}

## End(Not run)
```

bdsubset_hdf5_dataset *Create Subset of HDF5 Dataset*

Description

Creates a new HDF5 dataset containing only the specified rows or columns from an existing dataset. This operation is memory efficient as it uses HDF5's hyperslab selection for direct disk-to-disk copying without loading the entire dataset into memory.

Usage

```
bdsubset_hdf5_dataset(
  filename,
  dataset_path,
  indices,
  select_rows = TRUE,
  new_group = "",
  new_name = "",
  overwrite = FALSE
)
```

Arguments

`filename`	Character string. Path to the HDF5 file
`dataset_path`	Character string. Path to the source dataset (e.g., "/group1/dataset1")
`indices`	Integer vector. Row or column indices to include (1-based, as per R convention)

`select_rows`	Logical. If TRUE, selects rows; if FALSE, selects columns (default: TRUE)
`new_group`	Character string. Target group for the new dataset (default: same as source)
`new_name`	Character string. Name for the new dataset (default: original_name + "_subset")
`overwrite`	Logical. Whether to overwrite destination if it exists (default: FALSE)

Details

This function provides an efficient way to create subsets of large HDF5 datasets without loading all data into memory. It uses HDF5's native hyperslab selection mechanism for optimal performance with big data.

Key features:

- Memory efficient - processes one row/column at a time
- Direct disk-to-disk copying using HDF5 hyperslab selection
- Preserves all dataset attributes and properties
- Works with datasets of any size
- Automatic creation of parent groups if needed
- Support for both row and column selection

Value

Logical. TRUE on success, FALSE on failure

Index Convention

Indices follow R's 1-based convention (first element is index 1), but are automatically converted to HDF5's 0-based indexing internally.

Performance

This function is designed for big data scenarios. Memory usage is minimal regardless of source dataset size, making it suitable for datasets that don't fit in memory.

Requirements

- The HDF5 file must exist and be accessible
- The source dataset must exist and contain numeric data
- Indices must be valid (within dataset dimensions)
- User must have read-write permissions on the file

Author(s)

BigDataStatMeth package authors

See Also

Other BigDataStatMeth HDF5 utilities: `bdmove_hdf5_dataset()`

Examples

```
## Not run:
# Select specific rows (e.g., rows 1, 3, 5, 10-15)
success <- bdsubset_dataset("data.h5",
                           dataset_path = "/matrix/data",
                           indices = c(1, 3, 5, 10:15),
                           select_rows = TRUE,
                           new_name = "selected_rows")

# Select specific columns
success <- bdsubset_dataset("data.h5",
                           dataset_path = "/matrix/data",
                           indices = c(2, 4, 6:10),
                           select_rows = FALSE,
                           new_group = "/filtered",
                           new_name = "selected_cols")

# Create subset in different group
success <- bdsubset_dataset("data.h5",
                           dataset_path = "/raw_data/matrix",
                           indices = 1:100,  # First 100 rows
                           select_rows = TRUE,
                           new_group = "/processed",
                           new_name = "top_100_rows")

# Extract specific samples for analysis
interesting_samples <- c(15, 23, 45, 67, 89, 123)
success <- bdsubset_dataset("data.h5",
                           dataset_path = "/experiments/results",
                           indices = interesting_samples,
                           select_rows = TRUE,
                           new_name = "analysis_subset")

## End(Not run)
```

---

bdSVD_hdf5 *Singular Value Decomposition for HDF5-Stored Matrices*

Description

Computes the Singular Value Decomposition (SVD) of a large matrix stored in an HDF5 file. The SVD decomposes a matrix A into a product A = UDV' where U and V are orthogonal matrices and D is a diagonal matrix containing the singular values.

Usage

```
bdSVD_hdf5(
  filename,
  group = NULL,
  dataset = NULL,
  k = 2L,
  q = 1L,
  bcenter = TRUE,
  bscale = TRUE,
  rankthreshold = 0,
  overwrite = NULL,
  method = NULL,
  threads = NULL
)
```

Arguments

`filename`	Character string. Path to the HDF5 file containing the input matrix.
`group`	Character string. Path to the group containing the input dataset.
`dataset`	Character string. Name of the input dataset to decompose.
`k`	Integer. Number of local SVDs to concatenate at each level (default = 2). Controls the trade-off between memory usage and computation speed.
`q`	Integer. Number of levels for SVD computation (default = 1). Higher values can improve accuracy but increase computation time.
`bcenter`	Logical. If TRUE (default), centers the data by subtracting column means.
`bscale`	Logical. If TRUE (default), scales the centered columns by their standard deviations or root mean square.
`rankthreshold`	Numeric. Threshold for determining matrix rank (default = 0). Must be between 0 and 0.1. Used to approximate rank for nearly singular matrices.
`overwrite`	Logical. If TRUE, allows overwriting existing results.
`method`	Character string. Computation method: • "auto": Automatically selects between "full" and "blocks" based on matrix size • "blocks": Uses block-based computation (recommended for large matrices) • "full": Performs direct computation without partitioning
`threads`	Integer. Number of threads for parallel computation.

Details

This function implements a block-based SVD algorithm suitable for large matrices that may not fit in memory. Key features include:

- Automatic method selection based on matrix size
- Block-based computation for large matrices
- Data centering and scaling options
- Parallel processing support
- Rank approximation through threshold

- Memory-efficient incremental algorithm

The implementation uses an incremental algorithm with two key parameters:

- k: number of local SVDs to concatenate at each level
- q: number of levels in the computation

Value

A list with the following elements:

fn Path to the HDF5 file

ds_d Path to the dataset containing singular values

ds_u Path to the dataset containing left singular vectors

ds_v Path to the dataset containing right singular vectors

References

- Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review, 53(2), 217-288.

See Also

- `bdPCA_hdf5` for Principal Component Analysis
- `bdQR_hdf5` for QR decomposition

Examples

```
## Not run:
# Create a sample large matrix in HDF5

library(BigDataStatMeth)
library(rhdf5)

# Create a sample large matrix in HDF5
A <- matrix(rnorm(10000), 1000, 10)

fn <- "test_temp.hdf5"
bdCreate_hdf5_matrix(filename = fn, object = A, group = "data", dataset = "matrix")

# Compute SVD with default parameters
res <- bdSVD_hdf5(fn, "data", "matrix")

# Compute SVD with custom parameters
res <- bdSVD_hdf5(fn, "data", "matrix",
           k = 4, q = 2,
           bcenter = TRUE, bscale = TRUE,
           method = "blocks",
           threads = 4)
```

```
# list contents
h5ls(res$fn)

# Extract the result from HDF5 (d)
result_d_hdf5 <- h5read(res$fn, res$ds_d)
result_d_hdf5

# Compute the same SVD in R
result_d_r <- svd(A)$d
result_d_r

# Compare both results (should be TRUE)
all.equal(result_d_hdf5, result_d_r)

# Remove file
if (file.exists(fn)) {
  file.remove(fn)
}


## End(Not run)
```

bdtCrossprod *Efficient Matrix Transposed Cross-Product Computation*

Description

Computes matrix transposed cross-products efficiently using block-based algorithms and optional parallel processing. Supports both single-matrix (XX') and two-matrix (XY') transposed cross-products.

Usage

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

Arguments

A	Numeric matrix. First input matrix.
B	Optional numeric matrix. If provided, computes XY' instead of XX'.
transposed	Logical. If TRUE, uses transposed input matrix.

`block_size`	Integer. Block size for computation. If NULL, uses optimal block size based on matrix dimensions and cache size.
`paral`	Logical. If TRUE, enables parallel computation.
`threads`	Integer. Number of threads for parallel computation. If NULL, uses all available threads.

Details

This function implements efficient transposed cross-product computation using block-based algorithms optimized for cache efficiency and memory usage. Key features:

- Operation modes:
  - Single matrix: Computes XX'
  - Two matrices: Computes XY'
- Performance optimizations:
  - Block-based computation for cache efficiency
  - Parallel processing for large matrices
  - Automatic block size selection
  - Memory-efficient implementation

The function automatically selects optimal computation strategies based on input size and available resources. For large matrices, block-based computation is used to improve cache utilization.

Value

Numeric matrix containing the transposed cross-product result.

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- Kumar, V. et al. (1994). Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publishing Company.

See Also

- `bdCrossprod` for standard cross-product
- `bdblockMult` for block-based matrix multiplication

Examples

```
library(BigDataStatMeth)

# Single matrix transposed cross-product
n <- 100
p <- 60
X <- matrix(rnorm(n*p), nrow=n, ncol=p)
res <- bdtCrossprod(X)
```

```
# Verify against base R
all.equal(tcrossprod(X), res)

# Two-matrix transposed cross-product
n <- 100
p <- 100
Y <- matrix(rnorm(n*p), nrow=n)
res <- bdtCrossprod(X, Y)

# Parallel computation
res_par <- bdtCrossprod(X, Y,
                        paral = TRUE,
                        threads = 4)
```

---

bdtCrossprod_hdf5 *Transposed cross product with HDF5 matrices*

Description

Performs optimized transposed cross product operations on matrices stored in HDF5 format. For a single matrix A, computes A * A^t. For two matrices A and B, computes A * B^t. Uses block-wise processing for memory efficiency.

Usage

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

Arguments

`filename`	String indicating the HDF5 file path
`group`	String indicating the input group containing matrix A
`A`	String specifying the dataset name for matrix A

`B`	Optional string specifying dataset name for matrix B. If NULL, performs A * A^t
`groupB`	Optional string indicating group containing matrix B. If NULL, uses same group as A
`block_size`	Optional integer specifying the block size for processing. Default is automatically determined based on matrix dimensions
`mixblock_size`	Optional integer for memory block size in parallel processing
`paral`	Optional boolean indicating whether to use parallel processing. Default is false
`threads`	Optional integer specifying number of threads for parallel processing. If NULL, uses maximum available threads
`outgroup`	Optional string specifying output group. Default is "OUTPUT"
`outdataset`	Optional string specifying output dataset name. Default is "tCrossProd_A_x_B"
`overwrite`	Optional boolean indicating whether to overwrite existing datasets. Default is false

Details

The function implements block-wise matrix multiplication to handle large matrices efficiently. Block size is automatically optimized based on:

- Available memory
- Matrix dimensions
- Whether parallel processing is enabled

For parallel processing:

- Uses OpenMP for thread management
- Implements cache-friendly block operations
- Provides automatic thread count optimization

Memory efficiency is achieved through:

- Block-wise reading and writing
- Minimal temporary storage
- Proper resource cleanup

Mathematical operations:

- For single matrix A: computes A * A^t
- For two matrices A, B: computes A * B^t
- Optimized for numerical stability

Value

A list containing the location of the transposed crossproduct result:

fn Character string. Path to the HDF5 file containing the result

ds Character string. Full dataset path to the transposed crossproduct result (A %% *t(A) or A %%* t(B)) within the HDF5 file

Examples

```
## Not run:
library(BigDataStatMeth)
library(rhdf5)

# Create test matrix
N <- 1000
M <- 1000
set.seed(555)
a <- matrix(rnorm(N*M), N, M)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", a, "INPUT", "A",
                     overwriteFile = TRUE)

# Compute transposed cross product
bdtCrossprod_hdf5("test.hdf5", "INPUT", "A",
                  outgroup = "OUTPUT",
                  outdataset = "result",
                  block_size = 1024,
                  paral = TRUE,
                  threads = 4)

## End(Not run)
```

`bdWriteDiagonal_hdf5` *Write Matrix Diagonal to HDF5*

Description

Updates the diagonal elements of a matrix stored in an HDF5 file.

Usage

```
bdWriteDiagonal_hdf5(diagonal, filename, group, dataset)
```

Arguments

`diagonal`	Numeric vector. New diagonal elements to write.
`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Path to the group containing the dataset.
`dataset`	Character string. Name of the dataset to modify.

Details

This function provides efficient diagonal modification capabilities with:

- Write features:
  - Direct diagonal access
  - Type checking and validation
  - Support for large matrices
- Implementation features:
  - Safe HDF5 file operations
  - Memory-efficient implementation
  - Comprehensive error handling
  - Type conversion support

The function validates input types and dimensions before modification.

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the diagonal elements written (group/dataset)

References

- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdgetDiagonal_hdf5` for reading diagonal elements
- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
## Not run:
library(BigDataStatMeth)

# Create test matrix
X <- matrix(rnorm(100), 10, 10)

# Save to HDF5
bdCreate_hdf5_matrix("test.hdf5", X, "data", "matrix1",
                     overwriteFile = TRUE)

# Create new diagonal
new_diag <- seq(1, 10)

# Update diagonal
bdWriteDiagonal_hdf5(new_diag, "test.hdf5", "data", "matrix1")
```

```
# Verify
diag_elements <- bdgetDiagonal_hdf5("test.hdf5", "data", "matrix1")
print(diag_elements)

# Cleanup
if (file.exists("test.hdf5")) {
  file.remove("test.hdf5")
}

## End(Not run)
```

bdWriteOppsiteTriangularMatrix_hdf5

Write Upper/Lower Triangular Matrix

Description

Creates a symmetric matrix by mirroring values from one triangular part to the other in an HDF5-stored matrix. This function modifies the matrix in-place, either copying the upper triangular values to the lower triangular part or vice versa.

Usage

```
bdWriteOppsiteTriangularMatrix_hdf5(
  filename,
  group,
  dataset,
  copytolower = NULL,
  elementsBlock = 1000000L
)
```

Arguments

filename	Character string specifying the path to an existing HDF5 file
group	Character string indicating the input group containing the dataset
dataset	Character string specifying the dataset to be modified
copytolower	Logical. If TRUE, copies upper triangular to lower triangular. If FALSE (default), copies lower triangular to upper triangular.
elementsBlock	Integer defining the maximum number of elements to process in each block. Default is 1,000,000. For matrices larger than 5000x5000, automatically adjusted to number of rows or columns * 2.

Details

This function provides an efficient way to create symmetric matrices from triangular data. It operates directly on HDF5 datasets using block processing for memory efficiency. The function:

- Validates that the input matrix is square
- Processes the matrix in blocks for memory efficiency
- Performs in-place modification of the dataset
- Preserves the original values in the source triangular part
- Supports both upper-to-lower and lower-to-upper mirroring

The implementation uses block processing to handle large matrices efficiently, making it suitable for big data applications. The block size can be adjusted based on available memory and performance requirements.

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

ds Character string with the full dataset path to the modified matrix. The opposite triangular part is written to the same input dataset, completing the symmetric matrix (group/dataset)

References

- Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations, 4th Edition. Johns Hopkins University Press.
- The HDF Group. (2000-2010). HDF5 User's Guide.

See Also

- `bdCreate_hdf5_matrix` for creating HDF5 matrices

Examples

```
library(BigDataStatMeth)

# Create a matrix with upper triangular values
X <- matrix(rnorm(100), 10, 10)
X.1 <- X
X[lower.tri(X)] <- 0

# Save to HDF5
bdCreate_hdf5_matrix("test_file.hdf5", X, "data", "X",
                     overwriteFile = TRUE,
                     overwriteDataset = FALSE,
                     unlimited = FALSE)

# Mirror upper triangular to lower
bdWriteOppsiteTriangularMatrix_hdf5(
  filename = "test_file.hdf5",
  group = "data",
  dataset = "X",
  copytolower = TRUE,
  elementsBlock = 10
)

# Create a matrix with lower triangular values
X <- X.1
X[upper.tri(X)] <- 0

# Add to HDF5 file
bdCreate_hdf5_matrix("test_file.hdf5", X, "data", "Y",
                     overwriteFile = FALSE,
                     overwriteDataset = FALSE,
                     unlimited = FALSE)

# Mirror lower triangular to upper
bdWriteOppsiteTriangularMatrix_hdf5(
  filename = "test_file.hdf5",
  group = "data",
  dataset = "Y",
  copytolower = FALSE,
  elementsBlock = 10
)

# Cleanup
if (file.exists("test_file.hdf5")) {
  file.remove("test_file.hdf5")
}
```

---

bdWrite_hdf5_dimnames *Write dimnames to an HDF5 dataset*

Description

Write row and/or column names metadata for an existing dataset in an HDF5 file. Empty vectors skip the corresponding dimnames.

Usage

```
bdWrite_hdf5_dimnames(filename, group, dataset, rownames, colnames)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string. Group containing the dataset.
`dataset`	Character string. Dataset name inside `group`.

`rownames`	Character vector of row names. Use `character(0)` to skip writing row names. If provided, length must equal nrow.
`colnames`	Character vector of column names. Use `character(0)` to skip writing column names. If provided, length must equal ncol.

Details

The dataset `group/dataset` must already exist. When non-empty, `rownames` and `colnames` lengths are validated against the dataset dimensions.

Value

List with components. If an error occurs, all string values are returned as empty strings (""):

fn Character string with the HDF5 filename

dsrows Character string with the full dataset path to the row names, stored as ".`dataset`_dimnames/1" within the specified group

dscols Character string with the full dataset path to the column names, stored as ".`dataset`_dimnames/2" within the specified group

Examples

```
## Not run:
bdWrite_hdf5_dimnames(
  filename = "test.h5",
  group = "MGCCA_IN",
  dataset = "X",
  rownames = paste0("r", seq_len(100)),
  colnames = paste0("c", seq_len(50))
)

# Skip column names:
bdWrite_hdf5_dimnames("test.h5", "MGCCA_IN", "X",
                      rownames = paste0("r", 1:100),
                      colnames = character(0))

## End(Not run)
```

`bd_wproduct` *Weighted matrix–vector products and cross-products*

Description

Compute weighted operations using a diagonal weight from `w`:

- `"xtwx"`: $X'diag(w)X$ (row weights; `length(w)` = `nrow(X)`)
- `"xwxt"`: $Xdiag(w)X'$ (column weights; `length(w)` = `ncol(X)`)

- `"xw"` : $X diag(w)$ (column scaling; `length(w) = ncol(X)`)
- `"wx"` : $diag(w) X$ (row scaling; `length(w) = nrow(X)`)

Inputs may be base numeric matrices .

Usage

```
bd_wproduct(X, w, op)
```

Arguments

`X`	Numeric matrix (n x p).
`w`	Numeric weight vector (length `n` or `p`), or a 1D matrix coerced to a vector.
`op`	Character string (case-insensitive): one of `"XtwX"`/`"xtwx"`, `"XwXt"`/`"xwxt"`, `"Xw"`/`"xw"`, `"wX"`/`"wx"`.

Details

`w` is interpreted as the diagonal of a weight matrix; its required length depends on the operation: rows for `"xtwx"` and `"wx"`, columns for `"xwxt"` and `"xw"`.

Value

Numeric matrix with dimensions depending on `op`: `p x p` for `"xtwx"`, `n x n` for `"xwxt"`, and `n x p` for `"xw"`/`"wx"`.

Examples

```
set.seed(1)
n <- 10; p <- 5
X <- matrix(rnorm(n * p), n, p)
u <- runif(n); w <- u * (1 - u)
bd_wproduct(X, w, "xtwx")  # p x p
bd_wproduct(X, w, "wx")    # n x p (row scaling)

v <- runif(p)
bd_wproduct(X, v, "xw")    # n x p (col scaling)
bd_wproduct(X, v, "xwxt")  # n x n
```

`BigDataStatMeth` *BigDataStatMeth package documentation*

Description

BigDataStatMeth package documentation

`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

```
data(cancer)
```

`colesterol` *Dataset colesterol*

Description

This dataset contains a dummy data for import dataset example

colesterol.csv

This data is used in bdImportTextFile_hdf5() function.

`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