Package: BigDataStatMeth (via r-universe)

October 17, 2026

Type Package

Title Scalable Statistical Computing with HDF5-Backed Matrices

Version 2.0.6

Date 2026-09-16

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, and QR decomposition) written in C++ and R, along with building blocks from which higher-level multivariate methods such as canonical correlation analysis can be constructed. 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, utils, R6

LinkingTo Rcpp, RcppEigen, Rhdf5lib

Suggests Matrix, BiocStyle, knitr, rmarkdown, ggplot2, MASS

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>

Config/pak/sysreqs make

Repository https://cran.r-universe.dev

Date/Publication 2026-09-17 01:31:50 UTC

RemoteUrl https://github.com/cran/BigDataStatMeth

RemoteRef HEAD

RemoteSha 09e2a42420df2f5f61c6672b4f468bbc1ab0e336

Contents

[.HDF5Matrix . . . 4
[<-.HDF5Matrix . . . 5
%*% . . . 6
apply_function . . . 7
as.data.frame.HDF5Matrix . . . 8
as.matrix.HDF5Matrix . . . 9
bd_wproduct . . . 11
bdapply_Function_hdf5 . . . 12
bdblockMult . . . 14
bdblockSubstract . . . 16
bdblockSum . . . 18
bdCorr_matrix . . . 20
bdCreate_hdf5_group . . . 21
bdCreate_hdf5_matrix . . . 22
bdCrossprod . . . 23
bdgetDatasetsList_hdf5 . . . 25
bdImportData_hdf5 . . . 26
bdImportTextFile_hdf5 . . . 28
bdmove_hdf5_dataset . . . 30
bdpseudoinv . . . 32
bdpseudoinv_hdf5 . . . 34
bdReduce_hdf5_dataset . . . 36
bdScalarwproduct . . . 37
bdtCrossprod . . . 38
bdWrite_hdf5_dimnames . . . 40
BigDataStatMeth . . . 41
can_allocate . . . 43
cancer . . . 44
cbind.HDF5Matrix . . . 45
chol.HDF5Matrix . . . 46
close.HDF5Matrix . . . 47
colesterol . . . 48
colMaxs . . . 49
colMeans . . . 50
colMins . . . 51
colSds . . . 53
colSums . . . 54

colVars . . . 55
cor . . . 56
cor.HDF5Matrix . . . 57
crossprod . . . 59
diag . . . 60
diag_op . . . 61
diag_scale . . . 62
diag<- . . . 63
dim.HDF5Matrix . . . 64
dimnames.HDF5Matrix . . . 64
dimnames<-.HDF5Matrix . . . 66
eigen . . . 66
filter_low_coverage . . . 67
filter_maf . . . 69
get_available_ram . . . 70
get_cpu_cores . . . 71
get_memory_thresholds . . . 72
get_recommended_threads . . . 74
get_total_ram . . . 75
hdf5_apply . . . 76
hdf5_close_all . . . 78
hdf5_close_file . . . 79
hdf5_create_matrix . . . 80
hdf5_import . . . 81
hdf5_import_multiple . . . 83
hdf5_matrix . . . 84
hdf5_reduce . . . 85
hdf5_remove . . . 86
HDF5Matrix-S3 . . . 87
HDF5Matrix-scalar-aggregations . . . 88
hdf5matrix_options . . . 89
impute_snps . . . 90
is_open . . . 92
length.HDF5Matrix . . . 93
list_datasets . . . 93
memory_info . . . 94
miRNA . . . 95
multiply_sparse . . . 96
multiply_sparse.HDF5Matrix . . . 97
object_size . . . 98
Ops.HDF5Matrix . . . 98
prcomp.HDF5Matrix . . . 100
print.HDF5Matrix . . . 102
print.HDF5PCA . . . 103
pseudoinverse . . . 103
qr . . . 105
qr.HDF5Matrix . . . 105
rbind.HDF5Matrix . . . 107

reduce 108
scale 109
sd 111
show_hdf5matrix_options 112
solve.HDF5Matrix 112
split.HDF5Matrix 113
split_dataset 115
str.HDF5Matrix 116
svd 117
svd.HDF5Matrix 118
svd_auto_threshold 121
sweep 121
sweep.HDF5Matrix 122
system_info 123
tcrossprod 124
var 125

Index **127**

`[.HDF5Matrix` *Subset an HDF5Matrix*

Description

Subset an HDF5Matrix

Usage

```
## S3 method for class 'HDF5Matrix'
x[i, j, drop = TRUE, ...]
```

Arguments

`x`	An `HDF5Matrix` object
`i`	Row indices: numeric, integer, logical, or missing
`j`	Column indices: numeric, integer, logical, or missing
`drop`	Logical, whether to drop dimensions for single row/column results (default `TRUE`)
`...`	Ignored

Details

All standard R indexing modes are supported:

- Contiguous ranges: `X[1:100, 1:50]`
- Non-contiguous: `X[c(1,3,5), c(2,4)]`
- Negative: `X[-c(1,2), ]` (all except rows 1 and 2)
- Logical: `X[row_mask, col_mask]`
- Missing: `X[, ]` (entire dataset)

Value

Numeric matrix, or vector when `drop = TRUE` and one dimension has length 1

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/matrix", data = matrix(rnorm(100), 10, 10))
X <- hdf5_matrix(tmp, "data/matrix")

X[1:5, 1:3]              # submatrix
X[1, ]                   # single row as vector
X[1, , drop = FALSE]     # single row as matrix
X[, 2]                   # single column as vector
X[-c(1, 10), ]           # all except first and last row
X[c(TRUE, FALSE), ]      # logical row index
X[, ]                    # entire dataset

X$close()
unlink(tmp)
```

`[<-.HDF5Matrix` *Subsetting assignment for HDF5Matrix objects*

Description

Subsetting assignment for HDF5Matrix objects

Usage

```
## S3 replacement method for class 'HDF5Matrix'
x[i, j, ...] <- value
```

Arguments

`x`	An `HDF5Matrix` object
`i`	Row indices (numeric, logical, or missing)
`j`	Column indices (numeric, logical, or missing)
`...`	Ignored
`value`	Values to assign (scalar, vector, or matrix)

Details

Writes data to the HDF5 dataset backing the `HDF5Matrix` object. Supports:

- Scalar assignment: `X[i, j] <- 5`
- Vector assignment: `X[i, ] <- c(1, 2, 3)`
- Matrix assignment: `X[1:3, 1:3] <- matrix(...)`
- Full replacement: `X[] <- matrix(...)`

The value is automatically recycled or reshaped to match the target dimensions. Changes are written immediately to disk.

Value

The modified `HDF5Matrix` object (invisibly)

Examples

```
tmp <- tempfile(fileext = ".h5")

# Create a matrix
X  <- hdf5_create_matrix(tmp, "data/X", data = matrix(rnorm(100), 10, 10))

X <- hdf5_matrix(tmp, "data/X")

# Assign scalar
X[1, 1] <- 42

# Assign row
X[2, ] <- 1:10

# Assign block
X[1:3, 1:3] <- matrix(0, 3, 3)

hdf5_close_all()
unlink(tmp)
```

%*% *Matrix multiplication for HDF5Matrix*

Description

S3 generic for `%*%`. Dispatches to `%*%.HDF5Matrix` for `HDF5Matrix` objects, and to `base::%*%` for all others.

Usage

```
x %*% y
```

Arguments

x	Left-hand side matrix.
y	Right-hand side matrix.

Value

Result matrix.

---

apply_function *Apply a statistical or algebraic function to HDF5 datasets (generic)*

Description

Generic S3 function that applies a predefined algebraic or statistical operation to one or more **named datasets within the HDF5 group** of `x`, writing the results to `out_group`.

This function is not equivalent to `base::apply()`. It does not apply an arbitrary R function row- or column-wise to the matrix data of `x`. Instead, it dispatches one of several built-in C++ operations (QR, cross-product, Cholesky, etc.) to a batch of datasets stored in the HDF5 file, which need not be open as `HDF5Matrix` objects.

Usage

```
apply_function(x, ...)
```

Arguments

x	An `HDF5Matrix`.
...	Additional arguments forwarded to the method.

Details

Two access patterns are available:

- `apply_function(x, datasets = c("A","B"), func = "QR")` — use when you have an open `HDF5Matrix`; `x` provides the file and group context.
- `hdf5_apply(filename, group, datasets, func)` — use when you only have the file path and group name.

Valid `func` values: `"QR"`, `"CrossProd"`, `"tCrossProd"`, `"invChol"`, `"blockmult"`, `"CrossProd_double"`, `"tCrossProd_double"`, `"solve"`, `"normalize"`, `"sdmean"`, `"descChol"`.

Value

Named list with elements `filename`, `out_group`, `func`, `datasets`.

See Also

hdf5_apply for the standalone file-path version; crossprod.HDF5Matrix, qr.HDF5Matrix for single-dataset S3 equivalents of the most common operations.

Examples

```
fn <- tempfile(fileext = ".h5")

# Create two datasets in the same group
hdf5_create_matrix(fn, "data/A", data = matrix(rnorm(50), 5, 10))
hdf5_create_matrix(fn, "data/B", data = matrix(rnorm(50), 5, 10))

# Apply CrossProd to all datasets in the group
X   <- hdf5_matrix(fn, "data/A")
res <- apply_function(X, func = "CrossProd", out_group = "RESULTS")

hdf5_close_all()
unlink(fn)
```

---

as.data.frame.HDF5Matrix

Convert HDF5Matrix to data.frame

Description

Reads entire HDF5 dataset into memory as a data.frame. **WARNING**: This loads all data into RAM.

Usage

```
## S3 method for class 'HDF5Matrix'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  force = FALSE,
  max_size_mb = NULL,
  ...
)
```

Arguments

x	An HDF5Matrix object
row.names	Logical or character vector. Row names to use.
optional	Logical. Passed to as.data.frame.

force	Logical. If TRUE, skip size warnings.
max_size_mb	Numeric. Maximum size in MB to convert without warning.
...	Additional arguments passed to as.data.frame

Details

First converts to matrix using as.matrix.HDF5Matrix (with same size checks), then to data.frame. All memory warnings apply.

Value

data.frame with data from HDF5 file

See Also

as.matrix.HDF5Matrix

Examples

```
fn <- tempfile(fileext = ".h5")
X <- hdf5_create_matrix(fn, "data/X", data = matrix(rnorm(500), 100, 5))
df <- as.data.frame(X)
hdf5_close_all()
unlink(fn)
```

---

as.matrix.HDF5Matrix *Convert HDF5Matrix to in-memory matrix*

Description

Reads entire HDF5 dataset into memory as a standard R matrix. **WARNING**: This loads all data into RAM. For large datasets (>1GB), this may cause memory exhaustion.

Usage

```
## S3 method for class 'HDF5Matrix'
as.matrix(x, force = FALSE, max_size_mb = NULL, ...)
```

Arguments

x	An HDF5Matrix object
force	Logical. If TRUE, skip size warnings. Default FALSE.
max_size_mb	Numeric. Maximum size in MB to convert without warning. Default is NULL (auto-detect based on system RAM). Use Inf to disable.
...	Additional arguments (currently unused)

Details

Size thresholds and behavior:

Small datasets (< max_size_mb): Convert silently

Medium datasets (max_size_mb to 2GB): Show warning, require confirmation

Large datasets (> 2GB): Show error, require force=TRUE

Huge datasets (> 8GB): Block conversion even with force=TRUE

Memory estimation: The function estimates memory usage as: `nrow * ncol * 8 bytes (for numeric)`

Actual memory usage may be higher due to:

- R's internal overhead
- Temporary copies during conversion
- Other objects in memory

Recommendations:

- For large datasets, use subsetting instead: `X[1:1000, ]`
- For analysis, use HDF5Matrix methods directly (they work on-disk)
- Only convert to memory when absolutely necessary

Value

Standard R matrix with data from HDF5 file

See Also

`[.HDF5Matrix` for subsetting, `as.data.frame.HDF5Matrix` for data frame conversion

Examples

```
fn <- tempfile(fileext = ".h5")
X <- hdf5_create_matrix(fn, "data/X", data = matrix(rnorm(500), 100, 5))
mat <- as.matrix(X)
head(mat)

# Subsetting is more efficient for large datasets
subset <- X[1:10, 1:3]

hdf5_close_all()
unlink(fn)
```

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" : $Xdiag(w)$ (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
```

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

```
fn <- tempfile(fileext = ".h5")
Y <- matrix(rnorm(100), 10, 10)
X <- matrix(rnorm(100), 10, 10)
Z <- matrix(rnorm(100), 10, 10)

hdf5_create_matrix(fn, "data/Y", data = Y)
hdf5_create_matrix(fn, "data/X", data = X)
hdf5_create_matrix(fn, "data/Z", data = Z)

dsets <- list_datasets(fn, group = "data")

bdapply_Function_hdf5(filename = fn,
                      group = "data", datasets = dsets,
                      outgroup = "QR", func = "QR",
                      overwrite = TRUE)
hdf5_close_all()
unlink(fn)
```

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

```
# 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)
```

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

```
# 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)
```

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

```
# 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)
```

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

```
set.seed(123)
X <- matrix(rnorm(1000), nrow = 100, ncol = 10)

# Single matrix correlation
res <- bdCorr_matrix(X)

# Transposed (sample-sample correlations)
res_t <- bdCorr_matrix(X, trans_x = TRUE)

# Cross-correlation with a second matrix
Y <- matrix(rnorm(400), nrow = 100, ncol = 4)
res_xy <- bdCorr_matrix(X, Y)
```

---

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

hdf5_create_matrix.

Examples

```
fn <- tempfile(fileext = ".h5")
hdf5_create_matrix(fn, "tmp/seed", data = matrix(0, 1, 1))
bdCreate_hdf5_group(fn, "MGCCA_OUT/scores")
hdf5_close_all()
unlink(fn)
```

---

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

```
fn <- tempfile(fileext = ".h5")
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 = fn,
                     object = matA, group = "datasets",
                     dataset = "datasetA", transp = FALSE,
                     overwriteFile = TRUE,
                     overwriteDataset = TRUE,
                     unlimited = FALSE)
hdf5_close_all()
unlink(fn)
```

---

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

```
# 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)
all.equal(crossprod(X, Y), res)

# Parallel computation
res_par <- bdCrossprod(X, paral = TRUE, threads = 2)
```

`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 = NULL,
  prefix = NULL,
  recursive = FALSE
)
```

Arguments

`filename`	Character string. Path to the HDF5 file.
`group`	Character string or `NULL`. Group path within the HDF5 file. If `NULL` (default), the entire file is traversed recursively and dataset paths are returned relative to the root (e.g. `"INPUT/A"`, `"RESULTS/SVD/d"`).
`prefix`	Optional character string. Only return datasets whose name starts with this prefix.
`recursive`	Logical. If `TRUE`, recurse into subgroups and return full relative paths. Ignored when `group = NULL` (always recursive). Default `FALSE`.

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.

Examples

```
fn <- tempfile(fileext = ".h5")
X  <- hdf5_create_matrix(fn, "INPUT/A",  data = matrix(rnorm(100), 10, 10))
Y  <- hdf5_create_matrix(fn, "INPUT/B",  data = matrix(rnorm(100), 10, 10))
Z  <- hdf5_create_matrix(fn, "RESULTS/C",data = matrix(rnorm(100), 10, 10))

# All datasets in the file (recursive from root)
bdgetDatasetsList_hdf5(fn)

# Only datasets in INPUT group
bdgetDatasetsList_hdf5(fn, group = "INPUT")

# INPUT group, recursive (same result here, no subgroups)
bdgetDatasetsList_hdf5(fn, group = "INPUT", recursive = TRUE)

# Filter by prefix
bdgetDatasetsList_hdf5(fn, group = "INPUT", prefix = "A")

hdf5_close_all()
unlink(fn)
```

---

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

```
# Create a temporary CSV file to import
csv_file <- tempfile(fileext = ".csv")
hdf5_file <- tempfile(fileext = ".h5")
```

```
# Write sample data
data <- matrix(rnorm(50), nrow = 10, ncol = 5)
write.table(data, csv_file, sep = ",", row.names = FALSE, col.names = TRUE)

# Import CSV to HDF5
bdImportData_hdf5(
  inFile      = csv_file,
  destFile    = hdf5_file,
  destGroup   = "mydata",
  destDataset = "matrix1",
  header      = TRUE,
  sep         = ","
)

hdf5_close_all()
unlink(c(csv_file, hdf5_file))
```

`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

- hdf5_create_matrix for creating HDF5 matrices directly

Examples

```
hdf5_file <- tempfile(fileext = ".h5")
csv_file <- tempfile(fileext = ".csv")

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

# Import to HDF5
bdImportTextFile_hdf5(
  filename = csv_file,
  outputfile = hdf5_file,
  outGroup = "data",
  outDataset = "matrix1",
  sep = ",",
  header = TRUE,
  overwriteFile = TRUE
)

# Cleanup
unlink(c(csv_file, hdf5_file))
```

---

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

Examples

```
fn <- tempfile(fileext = ".h5")

# Create a dataset to move
hdf5_create_matrix(fn, "old_group/my_dataset",
                   data = matrix(rnorm(100), 10, 10))

# Move dataset to a different group
```

```
res <- bdmove_hdf5_dataset(fn,
                           source_path = "old_group/my_dataset",
                           dest_path   = "new_group/my_dataset")

# Rename dataset within the same group
hdf5_create_matrix(fn, "data/old_name",
                   data = matrix(rnorm(100), 10, 10))
res <- bdmove_hdf5_dataset(fn,
                           source_path = "data/old_name",
                           dest_path   = "data/new_name")

hdf5_close_all()
unlink(fn)
```

---

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.

Examples

```
# 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 and writes the result back to HDF5.

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 reads the complete input dataset into memory, computes the pseudoinverse via a direct LAPACK SVD, and writes the full result to HDF5. The computation is HDF5-backed for input and output, but it is not block-wise: both the input matrix and its pseudoinverse must fit in available RAM during the computation. For matrices that exceed available memory, this function is not currently suitable.

Key features:

- HDF5 Integration:
  - Reads the complete input matrix in a single call
  - Direct output to HDF5 format
- Implementation Features:
  - SVD-based computation via direct LAPACK calls

  - Parallel processing support for the singular-value scaling step
  - Flexible output options

The function handles:

- Data validation
- 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

```
fn <- tempfile(fileext = ".h5")
X <- matrix(c(1,2,3,2,4,6), 2, 3)
hdf5_create_matrix(fn, "data/X", data = X)

bdpseudoinv_hdf5(filename = fn,
                 group = "data",
                 dataset = "X",
                 outgroup = "results",
                 outdataset = "X_pinv",
                 overwrite = TRUE)
hdf5_close_all()
unlink(fn)
```

`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.

Examples

```
fn <- tempfile(fileext = ".h5")
hdf5_create_matrix(fn, "data/matrix1", data = matrix(1:100, 10, 10))
hdf5_create_matrix(fn, "data/matrix2", data = matrix(101:200, 10, 10))
hdf5_create_matrix(fn, "data/matrix3", data = matrix(201:300, 10, 10))

bdReduce_hdf5_dataset(
  filename   = fn,
  group      = "data",
  reducefunction = "+",
  outgroup   = "results",
  outdataset = "sum_matrix",
  overwrite  = TRUE
)
hdf5_close_all()
unlink(fn)
```

---

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)
```

---

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

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

```
# Two-matrix transposed cross-product
# Both matrices must have the same number of columns
n <- 100
p <- 60
Y <- matrix(rnorm(n*p), nrow=n, ncol=p)
res <- bdtCrossprod(X, Y)
all.equal(tcrossprod(X, Y), res)

# Parallel computation
res_par <- bdtCrossprod(X, paral = TRUE, threads = 2)
```

`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

```
fn <- tempfile(fileext = ".h5")
hdf5_create_matrix(fn, "MGCCA_IN/X",
                   data = matrix(rnorm(5000), 100, 50))

bdWrite_hdf5_dimnames(
  filename = fn,
  group    = "MGCCA_IN",
  dataset  = "X",
  rownames = paste0("r", seq_len(100)),
  colnames = paste0("c", seq_len(50))
)
hdf5_close_all()
unlink(fn)
```

BigDataStatMeth *BigDataStatMeth: Scalable statistical computing with R, C++, and HDF5*

Description

BigDataStatMeth provides statistical and linear algebra operations for matrices stored in HDF5 files. The package is designed for workflows in which matrices may be too large to be held entirely in memory, while still allowing users to work with familiar R functions.

The recommended user-facing interface is based on `HDF5Matrix` objects and standard R methods. HDF5-backed matrices can be manipulated using calls such as `dim()`, `[`, `%*%`, `crossprod()`, `tcrossprod()`, `scale()`, `cor()`, `svd()`, `prcomp()`, `qr()`, `chol()`, and `solve()`.

Main user-facing functionality

- Core HDF5 matrix handling: `hdf5_create_matrix()`, `hdf5_matrix()`, `list_datasets()`, `is_open()`, `close()`, and `hdf5_close_all()`.
- Subsetting and conversion: `[`, `[<-`, `as.matrix()`, and `as.data.frame()`.
- Dimension names: `rownames()`, `colnames()`, and `dimnames()`.
- Element-wise arithmetic: `+`, `-`, `*`, and `/` for `HDF5Matrix` objects.
- Matrix algebra: `%*%`, `crossprod()`, `tcrossprod()`, `cbind()`, and `rbind()`.
- Aggregations and summaries: `colSums()`, `rowSums()`, `colMeans()`, `rowMeans()`, `colVars()`, `rowVars()`, `colSds()`, `rowSds()`, `colMins()`, `rowMins()`, `colMaxs()`, `rowMaxs()`, `mean()`, `var()`, and `sd()`.
- Statistical transformations: `scale()`, `sweep()`, and `cor()`.
- Matrix decompositions and factorizations: `svd()`, `prcomp()`, `qr()`, `chol()`, `solve()`, `eigen()`, and `pseudoinverse()`.
- Diagonal, split, reduce, and apply operations: `diag()`, `diag_op()`, `diag_scale()`, `split_dataset()`, `reduce()`, and `apply_function()`.

Additional high-level utilities

Most user workflows can be expressed through `HDF5Matrix` objects and standard R methods. Some functions keep the `bd*` prefix because they provide additional utilities that do not map directly to a standard R generic, or because they expose workflows available in earlier versions of the package. Examples include utilities for creating HDF5 groups, moving datasets, and writing HDF5-backed dimension names. These functions remain part of the package API and are documented in their corresponding help pages.

Global options and HDF5 resources

Block-wise operations can be configured with `hdf5matrix_options()`, including options for parallel execution, number of threads, block size, and HDF5 compression. Open HDF5 resources can be closed explicitly with `close()` for individual objects or `hdf5_close_all()` for all handles tracked by the package.

Architecture and developer interfaces

BigDataStatMeth is organized around a standard R interface backed by a C++ computational infrastructure. The user-facing layer is based on `HDF5Matrix` objects and S3 methods, allowing HDF5-backed matrices to be used with familiar R functions.

Internally, a lightweight R6 layer connects these R methods with the C++ backend. The C++ infrastructure provides classes for managing HDF5 files, groups, and datasets, together with block-wise routines for linear algebra and statistical operations.

This design allows developers to implement new scalable methods from Rcpp-based code while reusing the package machinery for HDF5 file management, block iteration, compression handling, and numerical computation.

Getting started

See `vignette("BigDataStatMeth")` for a practical introduction to HDF5-backed matrices and the main user-facing functionality.

Examples

```
h5file <- tempfile(fileext = ".h5")

set.seed(1)
X <- matrix(rnorm(100 * 20), nrow = 100, ncol = 20)

X_h5 <- hdf5_create_matrix(
  filename = h5file,
  dataset = "data/X",
  data = X,
  overwrite = TRUE
)

dim(X_h5)
colMeans(X_h5)
```

```
XtX_h5 <- crossprod(X_h5)
dim(XtX_h5)

close(X_h5)
close(XtX_h5)
hdf5_close_all(verbose = FALSE)
```

`can_allocate` *Check if memory allocation is safe*

Description

Checks whether a given amount of memory can be safely allocated while maintaining a safety margin.

Usage

```
can_allocate(size_gb, safety_margin_pct = 20)
```

Arguments

`size_gb`	Size in gigabytes (GB) to check
`safety_margin_pct`	Percentage of available RAM to keep free (default 20 percent)

Details

This function checks if the requested memory can be allocated while keeping a safety margin of free RAM. This helps prevent:

- System instability from memory exhaustion
- Swapping (which degrades performance)
- Out-of-memory errors from other processes

Formula: `can_allocate = (size_gb < available_ram * (1 - safety_margin / 100))`

Safety margin guidelines:

- 20 percent (default): Conservative, recommended for most cases
- 10 percent: Moderate, for controlled environments
- 5 percent: Aggressive, only if you know what you're doing
- 0 percent: Maximum risk, not recommended

Value

Logical. TRUE if allocation is likely safe, FALSE otherwise

Note

This is a heuristic check, not a guarantee. Allocation can still fail due to memory fragmentation or competing processes.

See Also

get_available_ram

Examples

```
# Check if 1 GB can be safely allocated
if (can_allocate(1)) {
  message("1 GB allocation is safe")
} else {
  message("Not enough RAM for 1 GB allocation")
}

# Use it to decide how much data to load
fn <- tempfile(fileext = ".h5")
X  <- hdf5_create_matrix(fn, "data/M",
                         data = matrix(rnorm(1000), 100, 10))

size_gb <- prod(dim(X)) * 8 / 1e9   # estimate in GB
if (can_allocate(size_gb)) {
  mat <- as.matrix(X)
} else {
  mat <- X[1:50, ]   # load subset
}

hdf5_close_all()
unlink(fn)
```

---

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)
```

`cbind.HDF5Matrix` *Column-bind HDF5Matrix objects*

Description

Binds two or more `HDF5Matrix` objects by columns (appending columns to the right). All matrices must have the same number of rows. The operation is performed block-wise on disk.

Usage

```
## S3 method for class 'HDF5Matrix'
cbind(
  ...,
  deparse.level = 1,
  out_file = NULL,
  out_group = NULL,
  out_dataset = NULL,
  block_rows = 1000L,
  overwrite = FALSE,
  compression = NULL
)
```

Arguments

`...`	One or more `HDF5Matrix` objects (all with the same number of rows). Every argument must be an `HDF5Matrix`; plain R matrices are not accepted – convert them first with `hdf5_matrix` or `hdf5_create_matrix`.
`deparse.level`	Ignored (for S3 compatibility with base::cbind).
`out_file`	Output HDF5 file. `NULL` = same file as first argument.
`out_group`	Output group. `NULL` = `"BIND"`.
`out_dataset`	Output dataset name. `NULL` = auto-generated: for two inputs the name is `"A_cbind_B"`; for three or more inputs it is `"cbind_N"` where `N` is the number of inputs, to prevent unbounded name growth.
`block_rows`	Integer. Rows per I/O block (default 1000).
`overwrite`	Logical. Overwrite existing output. Default `FALSE`.
`compression`	Integer (0-9) or NULL. gzip compression level for the result datasets. NULL uses the global option set by `hdf5matrix_options` (default 1). Use `0` to disable compression (faster for benchmarks).

Value

`HDF5Matrix` pointing to the combined dataset.

Examples

```
fn <- tempfile(fileext = ".h5")

A  <- hdf5_create_matrix(fn, "grp/A", data = matrix(rnorm(100), 10, 10))
B  <- hdf5_create_matrix(fn, "grp/B", data = matrix(rnorm(100), 10, 10))

A <- hdf5_matrix(fn, "grp/A")
B <- hdf5_matrix(fn, "grp/B")
C <- cbind(A, B)          # columns of A followed by columns of B
dim(C)                    # nrow(A) x (ncol(A) + ncol(B))

hdf5_close_all()
unlink(fn)
```

`chol.HDF5Matrix` *Cholesky decomposition of a symmetric positive-definite HDF5Matrix*

Description

Computes the Cholesky factor and returns it as an upper-triangular matrix U such that `t(U) %*% U == A`, consistent with `base::chol()`. The internal C++ algorithm computes the lower-triangular factor L (A = L L'); the R/HDF5 transposition convention means the user receives L' = U. The input matrix must be square and symmetric positive-definite.

Usage

```
## S3 method for class 'HDF5Matrix'
chol(
  x,
  full_matrix = FALSE,
  overwrite = FALSE,
  threads = -1L,
  block_size = NULL,
  compression = NULL,
  ...
)
```

Arguments

`x`	An `HDF5Matrix`.
`full_matrix`	Logical. Return full symmetric matrix (L + L'). Default `FALSE`.
`overwrite`	Logical. Overwrite existing result. Default `FALSE`.
`threads`	Integer. OpenMP threads (-1 = auto).

block_size	Integer or NULL. Elements per block. NULL = auto.
compression	Integer (0-9) or NULL. gzip compression level for the result dataset. NULL uses the global option set by hdf5matrix_options (default 1). Use 0 to disable compression (faster for benchmarks).
...	Ignored (for S3 compatibility).

Value

HDF5Matrix containing the upper-triangular Cholesky factor U such that t(U) %*% U == A (consistent with base::chol()).

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/X", data = matrix(rnorm(10000), 100, 100))

# Create a symmetric positive-definite matrix: A = t(X) %*% X
X  <- hdf5_matrix(tmp, "data/X")
AtA <- crossprod(X)              # HDF5Matrix, square SPD
L   <- chol(AtA)

hdf5_close_all()
unlink(tmp)
```

---

close.HDF5Matrix *Close HDF5Matrix*

Description

Close an HDF5Matrix object and release file resources immediately. This is the standard R interface for resource cleanup.

Usage

```
## S3 method for class 'HDF5Matrix'
close(con, ...)
```

Arguments

con	An HDF5Matrix object
...	Additional arguments (currently ignored)

Details

Closes the HDF5 dataset without waiting for garbage collection. After calling `close()`, `is_valid()` returns `FALSE` and any further operations on the object will fail. The file is immediately accessible by other tools such as HDFView.

Both syntaxes work:

- `close(X)` - Standard R generic (recommended)
- `X$close()` - R6 method (still supported)

For emergency closure of all open HDF5 objects in the session, see `hdf5_close_all`.

Value

Invisible `NULL`

See Also

`hdf5_matrix` for opening datasets, `hdf5_close_all` for closing all open objects

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/matrix", data = matrix(rnorm(100), 10, 10))

# Open matrix
X <- hdf5_matrix(tmp, "data/matrix")
data <- X[1:5, 1:5]

# Close using S3 method (recommended)
close(X)

# Or using R6 method (still works)
# X$close()

X$is_valid()  # FALSE

unlink(tmp)
```

---

colesterol *Dataset colesterol*

Description

This dataset contains a dummy data for import dataset example

colesterol.csv

This data is used in bdImportTextFile_hdf5() function.

`colMaxs` *Column and row maximums for HDF5Matrix*

Description

Block-wise computation of column and row maximums.

Usage

```
colMaxs(x, ...)

## S3 method for class 'HDF5Matrix'
colMaxs(
  x,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)

rowMaxs(x, ...)

## S3 method for class 'HDF5Matrix'
rowMaxs(
  x,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

`x`	An `HDF5Matrix` object.
`...`	Ignored.
`paral`	Logical or NULL. Enable OpenMP parallelisation.
`wsize`	Integer or NULL. Block size for HDF5 reads (NULL = auto).
`threads`	Integer or NULL. Number of OpenMP threads (NULL = auto).

save_to	Where to save the result (see Details). NULL returns a plain R vector; a character string "group/dataset" saves in the same file as x; a named list list(file = "f.h5", path = "group/dataset") saves in a different file. In both non-NULL cases an HDF5Matrix handle is returned.
overwrite	Logical. Overwrite an existing dataset at save_to? (Currently always TRUE when save_to is not NULL.)

Value

A numeric vector (when save_to = NULL) or an HDF5Matrix handle to the persisted result.

---

colMeans *Column and row means for HDF5Matrix*

Description

Block-wise computation of column and row means without loading the full matrix into RAM.

Usage

```
colMeans(x, na.rm = FALSE, dims = 1L, ...)

rowMeans(x, na.rm = FALSE, dims = 1L, ...)

## S3 method for class 'HDF5Matrix'
colMeans(
  x,
  na.rm = FALSE,
  dims = 1,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)

## S3 method for class 'HDF5Matrix'
rowMeans(
  x,
  na.rm = FALSE,
  dims = 1,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
```

```
  ...
)
```

Arguments

x	An HDF5Matrix object.
na.rm	Ignored (included for compatibility with the base generic).
dims	Ignored (included for compatibility with the base generic).
...	Ignored.
paral	Logical or NULL. Enable OpenMP parallelisation.
wsize	Integer or NULL. Block size for HDF5 reads (NULL = auto).
threads	Integer or NULL. Number of OpenMP threads (NULL = auto).
save_to	Where to save the result (see Details). NULL returns a plain R vector; a character string "group/dataset" saves in the same file as x; a named list list(file = "f.h5", path = "group/dataset") saves in a different file. In both non-NULL cases an HDF5Matrix handle is returned.
overwrite	Logical. Overwrite an existing dataset at save_to? (Currently always TRUE when save_to is not NULL.)

Value

A numeric vector (when save_to = NULL) or an HDF5Matrix handle to the persisted result.

Examples

```
tmp <- tempfile(fileext = ".h5")
X <- hdf5_create_matrix(tmp, "data/M", data = matrix(rnorm(200), 20, 10))
cm <- colMeans(X)
hdf5_close_all()
unlink(tmp)
```

---

colMins *Column and row minimums for HDF5Matrix*

Description

Block-wise computation of column and row minimums.

Usage

```
colMins(x, ...)

## S3 method for class 'HDF5Matrix'
colMins(
  x,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)

rowMins(x, ...)

## S3 method for class 'HDF5Matrix'
rowMins(
  x,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

x	An HDF5Matrix object.
...	Ignored.
paral	Logical or NULL. Enable OpenMP parallelisation.
wsize	Integer or NULL. Block size for HDF5 reads (NULL = auto).
threads	Integer or NULL. Number of OpenMP threads (NULL = auto).
save_to	Where to save the result (see Details). NULL returns a plain R vector; a character string "group/dataset" saves in the same file as x; a named list list(file = "f.h5", path = "group/dataset") saves in a different file. In both non-NULL cases an HDF5Matrix handle is returned.
overwrite	Logical. Overwrite an existing dataset at save_to? (Currently always TRUE when save_to is not NULL.)

Value

A numeric vector (when save_to = NULL) or an HDF5Matrix handle to the persisted result.

colSds *Column and row standard deviations for HDF5Matrix*

Description

Block-wise computation of column and row standard deviations (Bessel's correction, n-1).

Usage

```
colSds(x, ...)

## S3 method for class 'HDF5Matrix'
colSds(
  x,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)

rowSds(x, ...)

## S3 method for class 'HDF5Matrix'
rowSds(
  x,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

x	An HDF5Matrix object.
...	Ignored.
paral	Logical or NULL. Enable OpenMP parallelisation.
wsize	Integer or NULL. Block size for HDF5 reads (NULL = auto).
threads	Integer or NULL. Number of OpenMP threads (NULL = auto).
save_to	Where to save the result (see Details). NULL returns a plain R vector; a character string "group/dataset" saves in the same file as x; a named list list(file = "f.h5", path = "group/dataset") saves in a different file. In both non-NULL cases an HDF5Matrix handle is returned.

`overwrite` Logical. Overwrite an existing dataset at `save_to`? (Currently always TRUE when `save_to` is not NULL.)

Value

A numeric vector (when `save_to = NULL`) or an `HDF5Matrix` handle to the persisted result.

`colSums` *Column and row sums for HDF5Matrix*

Description

Block-wise computation of column and row sums without loading the full matrix into RAM. Results can optionally be persisted to an HDF5 file.

Usage

```
colSums(x, na.rm = FALSE, dims = 1L, ...)

rowSums(x, na.rm = FALSE, dims = 1L, ...)

## S3 method for class 'HDF5Matrix'
colSums(
  x,
  na.rm = FALSE,
  dims = 1,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)

## S3 method for class 'HDF5Matrix'
rowSums(
  x,
  na.rm = FALSE,
  dims = 1,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

x	An HDF5Matrix object.
na.rm	Ignored (included for compatibility with the base generic).
dims	Ignored (included for compatibility with the base generic).
...	Ignored.
paral	Logical or NULL. Enable OpenMP parallelisation.
wsize	Integer or NULL. Block size for HDF5 reads (NULL = auto).
threads	Integer or NULL. Number of OpenMP threads (NULL = auto).
save_to	Where to save the result (see Details). NULL returns a plain R vector; a character string "group/dataset" saves in the same file as x; a named list list(file = "f.h5", path = "group/dataset") saves in a different file. In both non-NULL cases an HDF5Matrix handle is returned.
overwrite	Logical. Overwrite an existing dataset at save_to? (Currently always TRUE when save_to is not NULL.)

Value

A numeric vector (when save_to = NULL) or an HDF5Matrix handle to the persisted result.

Examples

```
tmp <- tempfile(fileext = ".h5")
X <- hdf5_create_matrix(tmp, "data/M", data = matrix(rnorm(200), 20, 10))
cs <- colSums(X)         # R vector
hdf5_close_all()
unlink(tmp)
```

---

colVars *Column and row variances for HDF5Matrix*

Description

Block-wise computation of column and row variances (Bessel's correction, n-1). Returns NaN for columns/rows with fewer than 2 observations, matching base R behaviour.

Usage

```
colVars(x, ...)

## S3 method for class 'HDF5Matrix'
colVars(
  x,
  paral = NULL,
```

```
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)

rowVars(x, ...)

## S3 method for class 'HDF5Matrix'
rowVars(
  x,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

x	An HDF5Matrix object.
...	Ignored.
paral	Logical or NULL. Enable OpenMP parallelisation.
wsize	Integer or NULL. Block size for HDF5 reads (NULL = auto).
threads	Integer or NULL. Number of OpenMP threads (NULL = auto).
save_to	Where to save the result (see Details). NULL returns a plain R vector; a character string "group/dataset" saves in the same file as x; a named list list(file = "f.h5", path = "group/dataset") saves in a different file. In both non-NULL cases an HDF5Matrix handle is returned.
overwrite	Logical. Overwrite an existing dataset at save_to? (Currently always TRUE when save_to is not NULL.)

Value

A numeric vector (when save_to = NULL) or an HDF5Matrix handle to the persisted result.

---

cor *Correlation (generic)*

Description

S3 generic for cor(). Dispatches to cor.HDF5Matrix for HDF5Matrix objects, and to stats::cor() for all others.

Usage

```
cor(x, y = NULL, use = "everything", method = "pearson", ...)
```

Arguments

x	A matrix or HDF5Matrix object.
y	Optional second matrix.
use	Character. Method for handling missing values.
method	Character. Correlation method: "pearson" or "spearman".
...	Additional arguments passed to the method.

Value

Correlation matrix.

---

cor.HDF5Matrix *Correlation matrix for HDF5Matrix objects*

Description

Block-wise computation of Pearson or Spearman correlation, running entirely on disk without loading the full matrix into RAM. Supports both auto-correlation cor(X) and cross-correlation cor(X, Y).

Usage

```
## S3 method for class 'HDF5Matrix'
cor(
  x,
  y = NULL,
  use = "everything",
  method = "pearson",
  trans_x = FALSE,
  trans_y = FALSE,
  compute_pvalues = TRUE,
  block_size = NULL,
  threads = NULL,
  result_path = NULL,
  compression = NULL,
  ...
)
```

Arguments

x	An HDF5Matrix object.
y	An HDF5Matrix for cross-correlation, or NULL (default) to compute cor(x, x).
use	Character string. Only "everything" (default) and "complete.obs" are currently supported.
method	"pearson" (default) or "spearman".
trans_x	Logical. If TRUE, correlate rows of x instead of columns (useful for sample-sample correlations in omics data). Default FALSE.
trans_y	Logical. Same for y. Default FALSE.
compute_pvalues	Logical. Also compute and store p-values on disk. Default TRUE.
block_size	Integer or NULL. Block size for HDF5 reads (NULL = auto).
threads	Integer or NULL. Number of OpenMP threads (NULL = auto).
result_path	Output location: NULL (default) writes to "CORR/<dataset>/correlation" in the same file as x. A character string specifies a custom output group in the same file. A named list list(file=, group=) writes to a different file.
compression	Integer (0-9) or NULL. gzip compression level for the result datasets. NULL uses the global option set by hdf5matrix_options (default 1). Use 0 to disable compression (faster for benchmarks).
...	Ignored (for S3 compatibility).

Value

An HDF5Matrix pointing to the correlation matrix on disk. Attributes attached to the result:

cor.method The correlation method used.

cor.type "single" or "cross".

cor.n.vars Number of variables (columns/rows correlated).

cor.n.obs Number of observations used.

cor.pvalues.path HDF5 path to the p-values dataset (present only when compute_pvalues = TRUE).

Examples

```
tmp <- tempfile(fileext = ".h5")
X   <- hdf5_create_matrix(tmp, "data/X",
                          data = matrix(rnorm(500), 50, 10))

# Auto-correlation: cor(X) — 10 x 10 matrix
C <- cor(X)
dim(C)
cat("method:", attr(C, "cor.method"), "\n")

# Spearman
Cs <- cor(X, method = "spearman")
```

```
dim(Cs)

# Sample-sample correlation (rows)
Sr <- cor(X, trans_x = TRUE)   # 50 x 50
dim(Sr)

X$close(); C$close(); Cs$close(); Sr$close()
unlink(tmp)
```

`crossprod` *Cross product of HDF5Matrix objects*

Description

S3 generic for `crossprod()`. Dispatches to `crossprod.HDF5Matrix` for `HDF5Matrix` objects, and to `base::crossprod()` for all others.

Usage

```
crossprod(x, y = NULL, ...)

## S3 method for class 'HDF5Matrix'
crossprod(x, y = NULL, outgroup = NULL, outdataset = NULL, ...)
```

Arguments

`x`	An `HDF5Matrix` object.
`y`	An `HDF5Matrix` object, or `NULL` (default) to compute `t(x) %*% x`.
`...`	Ignored.
`outgroup`	Character or `NULL`. HDF5 group where the result is stored. Default `"OUTPUT"`.
`outdataset`	Character or `NULL`. Dataset name for the result. Default `"CrossProd_x"` (single matrix) or `"CrossProd_x_x_y"` (two matrices).

Details

Computes $t(x) \times y$ (or $t(x) \times x$ when `y = NULL`). Uses the dedicated BigDataStatMeth block-wise cross-product algorithm, which is more efficient than explicitly computing `t(x) %*% y`.

Performance settings:

This method uses global options set via `hdf5matrix_options`.

Symmetric optimization:

When `y = NULL` or `y` refers to the same dataset as `x`, the symmetric optimisation (`bisSymetric = TRUE`) is applied automatically, providing significant speedup.

Value

Result of the cross product.

A new HDF5Matrix pointing to the result dataset.

See Also

hdf5matrix_options for global performance settings

Examples

```
fn <- tempfile(fileext = ".h5")
X <- hdf5_create_matrix(fn, "INPUT/X", data = matrix(rnorm(60), 6, 10))
Y <- hdf5_create_matrix(fn, "INPUT/Y", data = matrix(rnorm(60), 6, 10))

# t(X) %*% X  → stored in OUTPUT/CrossProd_X
C1 <- crossprod(X)
dim(C1)

# t(X) %*% Y  → stored in OUTPUT/CrossProd_X_x_Y
C2 <- crossprod(X, Y)

# Custom output location
C3 <- crossprod(X, outgroup = "RESULTS", outdataset = "my_crossprod")

hdf5_close_all()
unlink(fn)
```

---

diag *Extract or construct a diagonal for HDF5Matrix*

Description

Overrides base::diag() to dispatch on HDF5Matrix objects. For plain R matrices/vectors the call is forwarded to base::diag().

When x is an HDF5Matrix, extracts the diagonal as an in-memory numeric vector (length = min(nrow, ncol)).

Usage

```
diag(x, ...)

## Default S3 method:
diag(x, nrow, ncol, names = TRUE, ...)

## S3 method for class 'HDF5Matrix'
diag(x, ...)
```

Arguments

x	An HDF5Matrix or any object accepted by base::diag().
...	Ignored.
nrow	Passed to base::diag() for non-HDF5Matrix objects.
ncol	Passed to base::diag() for non-HDF5Matrix objects.
names	Passed to base::diag() for non-HDF5Matrix objects.

Value

For HDF5Matrix: numeric vector of diagonal elements. For plain R objects: result of base::diag().

See Also

diag

---

diag_op *Diagonal-vector operation on an HDF5Matrix*

Description

Applies an element-wise binary operation between an HDF5Matrix and a diagonal vector (a 1-row or 1-column HDF5Matrix). The vector is broadcast across each row of the matrix.

The standard arithmetic operators (+, -, *, /) dispatch automatically to this function when one operand is a 1-row or 1-column HDF5Matrix.

Usage

```
diag_op(x, diag, op = "+", ...)

## S3 method for class 'HDF5Matrix'
diag_op(x, diag, op = "+", outgroup = NULL, outdataset = NULL, ...)
```

Arguments

x	An HDF5Matrix (matrix, $m \times n$).
diag	An HDF5Matrix with one row or one column.
op	Character. One of "+", "-", "*", "/".
...	Additional arguments passed to x$diag_op(): outgroup, outdataset, overwrite, threads, compression.
outgroup	Character or NULL. HDF5 group where the result is stored. Default "OUTPUT".
outdataset	Character or NULL. Dataset name for the result.

Value

A new HDF5Matrix.

See Also

diag_scale

Examples

```
tmp <- tempfile(fileext = ".h5")
M   <- hdf5_create_matrix(tmp, "data/M", data = matrix(rnorm(10000), 100, 100))
d   <- hdf5_create_matrix(tmp, "data/d", data = matrix(rnorm(10000), 100, 100))
R1  <- diag_op(M, d, "*")    # scale each column
R2  <- M * d                  # same via operator auto-dispatch
hdf5_close_all()
unlink(tmp)
```

---

diag_scale *Scalar diagonal operation on an HDF5Matrix*

Description

Applies a scalar arithmetic operation to the diagonal elements of an HDF5Matrix. Off-diagonal elements are not modified. Delegates to bdDiag_scalar_hdf5().

Usage

```
diag_scale(x, scalar, op = "multiply", ...)

## S3 method for class 'HDF5Matrix'
diag_scale(
  x,
  scalar,
  op = "multiply",
  outgroup = NULL,
  outdataset = NULL,
  overwrite = FALSE,
  ...
)
```

Arguments

x	An HDF5Matrix.
scalar	Numeric scalar.
op	Operation: "add", "subtract", "multiply" (default), or "divide".
...	Additional arguments passed to x$diag_scale(): outgroup, outdataset, overwrite, threads, compression.
outgroup	Character or NULL. HDF5 group where the result is stored. Default "OUTPUT".
outdataset	Character or NULL. Dataset name for the result.
overwrite	Logical. If TRUE, overwrite the existing result dataset. Default FALSE.

Value

A new HDF5Matrix.

See Also

diag_op

Examples

```
tmp <- tempfile(fileext = ".h5")
M   <- hdf5_create_matrix(tmp, "data/M", data = diag(5))
R   <- diag_scale(M, scalar = 3, op = "multiply")
hdf5_close_all()
unlink(tmp)
```

---

diag<- *Set diagonal of an HDF5Matrix (generic)*

Description

S3 generic for diag<-. Dispatches to the diag<-.HDF5Matrix method for HDF5Matrix objects, and to base::diag<- for all others.

Usage

```
diag(x) <- value
```

Arguments

x	A matrix or HDF5Matrix object.
value	Numeric vector of replacement values for the diagonal.

Value

The modified object with the diagonal replaced.

dim.HDF5Matrix *Dimensions of an HDF5Matrix*

Description

Dimensions of an HDF5Matrix

Usage

```
## S3 method for class 'HDF5Matrix'
dim(x)
```

Arguments

x	An HDF5Matrix object

Value

Integer vector c(nrows, ncols), or NULL if the dataset has been closed

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/matrix", data = matrix(rnorm(100), 10, 10))
X <- hdf5_matrix(tmp, "data/matrix")

dim(X)    # c(10, 10)
nrow(X)   # 10
ncol(X)   # 10

X$close()
unlink(tmp)
```

dimnames.HDF5Matrix *Get dimension names of an HDF5Matrix*

Description

Returns the row and column names stored alongside the HDF5 dataset, following the BigDataStatMeth convention. Returns NULL when no names have been stored for a given dimension.

Usage

```
## S3 method for class 'HDF5Matrix'
dimnames(x)

rownames.HDF5Matrix(x, do.NULL = TRUE, prefix = "row")

colnames.HDF5Matrix(x, do.NULL = TRUE, prefix = "col")

## S3 replacement method for class 'HDF5Matrix'
rownames(x) <- value

## S3 replacement method for class 'HDF5Matrix'
colnames(x) <- value
```

Arguments

x	An HDF5Matrix object
do.NULL	Logical. Ignored; present for base compatibility.
prefix	Character. Ignored; present for base compatibility.
value	Character vector of column names, or NULL to remove.

Value

A list of length 2 with elements [[1]] (rownames) and [[2]] (colnames), or NULL for dimensions without names. Returns NULL when neither dimension has names.

Examples

```
tmp <- tempfile(fileext = ".h5")

m <- matrix(1:6, 2, 3,
            dimnames = list(c("r1","r2"), c("c1","c2","c3")))

X <- hdf5_create_matrix(tmp, "data/mat", data = m)

dimnames(X)
rownames(X)
colnames(X)
rownames(X) <- c("row1", "row2")
rownames(X)
hdf5_close_all()
unlink(tmp)
```

dimnames<-.HDF5Matrix *Set dimension names on an HDF5Matrix*

Description

Writes row and/or column names to the HDF5 file alongside the dataset. Setting a dimension name to NULL removes names for that dimension.

Usage

```
## S3 replacement method for class 'HDF5Matrix'
dimnames(x) <- value
```

Arguments

x	An HDF5Matrix object
value	A list of length 2: list(rownames, colnames). Either element may be NULL to skip that dimension.

Value

x invisibly

eigen *Spectral decomposition*

Description

Overrides base::eigen() to dispatch on HDF5Matrix objects. For plain R matrices the call is forwarded to base::eigen().

Usage

```
eigen(x, symmetric, ...)

## Default S3 method:
eigen(
  x,
  symmetric = !isSymmetric(x),
  only.values = FALSE,
  EISPACK = FALSE,
  ...
)

## S3 method for class 'HDF5Matrix'
eigen(x, symmetric = TRUE, ...)
```

Arguments

x	An HDF5Matrix or any R object accepted by base::eigen().
symmetric	Logical. Whether to assume x is symmetric. Passed to base::eigen() for plain R objects.
...	For HDF5Matrix: named arguments forwarded to x$eigen() — k (integer, number of eigenvalues), which (character, "LM"/"SM"/etc.), compute_vectors (logical), tolerance (numeric), max_iter (integer), overwrite (logical), threads (integer). Ignored for eigen.default.
only.values	Logical. Ignored; present for compatibility with base::eigen.
EISPACK	Logical. Ignored; present for compatibility with base::eigen.

Value

For HDF5Matrix: a named list with elements values (numeric vector), vectors (HDF5Matrix or NULL), values_imag, and is_symmetric. For other objects: the result of base::eigen().

See Also

svd.HDF5Matrix, prcomp.HDF5Matrix

Examples

```
tmp <- tempfile(fileext = ".h5")
m   <- crossprod(matrix(rnorm(400), 20, 20))
X   <- hdf5_create_matrix(tmp, "data/M", data = m)
ev  <- eigen(X, symmetric = TRUE, k = 5L)
ev$values
close(ev$vectors)
close(X)
unlink(tmp)
```

---

filter_low_coverage *Remove high-missingness features from an HDF5Matrix*

Description

Removes columns (SNPs) or rows (samples) that carry too many missing values. Writes the result to a new dataset.

Missing values are the entries equal to 3, the missing-data code of the 0/1/2/3 genotype encoding shared by the omics functions; NA or NaN entries are **not** counted, so recode them to 3 beforehand. A column (or row) is removed when its proportion of 3s is greater than or equal to pcent.

Row and column names of x, when present, are carried over to the result: the names of the filtered axis are subset to the surviving elements.

When out_group/out_dataset are NULL (default) the result is written alongside the input dataset with the suffix "_filtered".

Usage

```
filter_low_coverage(x, ...)

## S3 method for class 'HDF5Matrix'
filter_low_coverage(
  x,
  out_group = NULL,
  out_dataset = NULL,
  pcent = 0.05,
  by_cols = TRUE,
  overwrite = FALSE,
  ...
)
```

Arguments

x	An HDF5Matrix containing SNP data with missing values coded as 3.
...	Ignored.
out_group	Output group. NULL (default) = same group as input.
out_dataset	Output dataset name. NULL (default) = input name + "_filtered".
pcent	Numeric in [0,1]. Missing-data threshold (default 0.05). Features whose proportion of 3s reaches this value are removed.
by_cols	Logical. Filter columns (TRUE, default) or rows.
overwrite	Logical. Overwrite existing output. Default FALSE.

Value

HDF5Matrix pointing to the filtered dataset, carrying the dimension names of the surviving elements.

Examples

```
set.seed(42)
fn <- tempfile(fileext = ".h5")
# 0/1/2 genotypes with about 10% missing values, coded as 3
snps <- matrix(sample(c(0, 1, 2, 3), 200, replace = TRUE,
                      prob = c(.3, .3, .3, .1)), 20, 10)
X   <- hdf5_create_matrix(fn, "geno/raw", data = snps)

# Filter with auto output path (adds "_filtered" suffix)
out <- filter_low_coverage(X, pcent = 0.2)

# Filter with explicit output
out2 <- filter_low_coverage(X, pcent = 0.2, out_group = "geno",
                            out_dataset = "filtered", overwrite = TRUE)
hdf5_close_all()
unlink(fn)
```

`filter_maf` *Remove SNPs by Minor Allele Frequency from an HDF5Matrix*

Description

Removes the columns or rows whose Minor Allele Frequency (MAF) is at or below `maf_threshold`, the usual way of dropping rare variants. Designed for 0/1/2-coded diploid genotype matrices.

MAF is computed as `maf = n0/n + 0.5 * n1/n`, taking `1 - maf` when that value exceeds 0.5, where `n0` and `n1` count the 0s and 1s of the feature. Only features with `maf > maf_threshold` are kept.

Row and column names of `x`, when present, are carried over to the result: the names of the filtered axis are subset to the surviving elements.

When `out_group`/`out_dataset` are `NULL` (default) the result is written alongside the input dataset with suffix `"_maf_filtered"`.

Usage

```
filter_maf(x, ...)

## S3 method for class 'HDF5Matrix'
filter_maf(
  x,
  out_group = NULL,
  out_dataset = NULL,
  maf_threshold = 0.05,
  by_cols = FALSE,
  block_size = 100L,
  overwrite = FALSE,
  ...
)
```

Arguments

`x`	An `HDF5Matrix` containing SNP data.
`...`	Ignored.
`out_group`	Output group. `NULL` (default) = same group as input.
`out_dataset`	Output dataset name. `NULL` (default) = input name + `"_maf_filtered"`.
`maf_threshold`	Numeric in [0, 0.5]. MAF threshold (default `0.05`). SNPs with MAF at or **below** this value are removed.
`by_cols`	Logical. Treat SNPs as columns (`TRUE`) or as rows (`FALSE`, default).
`block_size`	Integer. Block size for I/O. Default `100L`.
`overwrite`	Logical. Overwrite existing output. Default `FALSE`.

Value

HDF5Matrix pointing to the filtered dataset, carrying the dimension names of the surviving elements.

Examples

```
set.seed(42)
fn <- tempfile(fileext = ".h5")
snps <- matrix(sample(c(0, 1, 2), 200, replace = TRUE,
                      prob = c(.6, .3, .1)), 20, 10)
X   <- hdf5_create_matrix(fn, "geno/raw", data = snps)

# Filter with auto output path (adds "_maf_filtered" suffix)
out <- filter_maf(X, maf_threshold = 0.05)

# Filter with explicit output
out2 <- filter_maf(X, out_group = "geno",
                   out_dataset = "maf_filtered", overwrite = TRUE)
hdf5_close_all()
unlink(fn)
```

---

get_available_ram *Get available (free) system RAM*

Description

Returns the amount of RAM currently available for allocation.

Usage

```
get_available_ram()
```

Details

This function returns the RAM that can be allocated without swapping. The value changes dynamically as processes allocate and free memory.

Important notes:

- Value can change rapidly; don't cache it
- On Linux, uses MemAvailable (more accurate than MemFree)
- Includes memory that can be reclaimed from caches
- Actual allocatable memory may be slightly less

Use case: Check available RAM before loading large datasets into memory.

Value

Numeric value with available RAM in gigabytes (GB)

See Also

get_total_ram, can_allocate

Examples

```
available <- get_available_ram()
cat("Available RAM:", round(available, 2), "GB\n")

# Use it to decide how much data to load
fn <- tempfile(fileext = ".h5")
X  <- hdf5_create_matrix(fn, "data/M",
                         data = matrix(rnorm(1000), 100, 10))

size_gb <- prod(dim(X)) * 8 / 1e9
if (get_available_ram() > size_gb * 1.2) {
  mat <- as.matrix(X)
} else {
  mat <- X[1:50, ]
}

hdf5_close_all()
unlink(fn)
```

---

get_cpu_cores *Get number of CPU cores*

Description

Returns the number of logical CPU cores (processors) available.

Usage

```
get_cpu_cores()
```

Details

This function returns the number of logical processors, which includes cores from hyperthreading/SMT. Useful for configuring parallel processing.

Typical values:

- 4-core CPU without hyperthreading: 4
- 4-core CPU with hyperthreading: 8

- 8-core CPU with hyperthreading: 16

Usage for parallelization: Don't blindly use all cores. A common practice is to use 80-90 percent of available cores to leave room for the OS and other processes.

Value

Integer with number of CPU cores

Note

- Returns logical cores (with hyperthreading), not physical cores
- On systems with CPU pinning, may return fewer cores
- Value reflects cores available to the process

See Also

get_total_ram

Examples

```
# Get CPU cores
cores <- get_cpu_cores()
cat("System has", cores, "CPU cores\n")

# Configure parallel processing (use 80 percent of cores)
threads <- max(1, floor(cores * 0.8))
options(BigDataStatMeth.threads = threads)
```

---

get_memory_thresholds *Get dynamic memory thresholds based on system RAM*

Description

Calculates appropriate memory thresholds for conversions based on total system RAM. Returns conservative values suitable for most systems.

Usage

```
get_memory_thresholds(total_ram = NULL)
```

Arguments

total_ram	Numeric. Total system RAM in GB. If NULL (default), auto-detects using C++ implementation.

Details

Calculation logic:

Uses percentage of total RAM with safety margins:

- Silent: 15\
- Warning: 30\
- Force: 50\
- Blocked: 80\

Fallback values (if RAM detection fails):

- Silent: 2 GB
- Warning: 4 GB
- Force: 8 GB
- Blocked: 16 GB

Value

Named list with thresholds in MB:

silent Size below which conversions happen silently

warning Size requiring user confirmation

force Size requiring force=TRUE

blocked Size that cannot be converted

See Also

`get_total_ram`, `memory_info`

Examples

```
# Auto-detect
thresholds <- get_memory_thresholds()
# $silent: 3000 MB, $warning: 8000 MB, etc.

# Manual specification (e.g., for 32GB system)
thresholds <- get_memory_thresholds(total_ram = 32)
```

`get_recommended_threads` *Get recommended number of threads for parallel operations*

Description

Returns a recommended number of threads for parallel operations, based on available CPU cores and system load.

Usage

```
get_recommended_threads(use_fraction = 0.8)
```

Arguments

`use_fraction`	Numeric. Fraction of available cores to use (default 0.8). Using all cores (1.0) may overload the system.

Details

This function uses the C++ implementation to detect CPU cores and returns a conservative estimate leaving room for the OS and other processes.

Default behavior (use_fraction = 0.8):

- 4 cores → 3 threads
- 8 cores → 6 threads
- 16 cores → 13 threads

Value

Integer with recommended number of threads (minimum 1)

See Also

`get_cpu_cores`

Examples

```
# Get recommended threads
threads <- get_recommended_threads()

# Use for OpenMP operations
options(BigDataStatMeth.threads = threads)

# More aggressive (use 90% of cores)
threads <- get_recommended_threads(use_fraction = 0.9)
```

`get_total_ram` *Get total system RAM*

Description

Returns the total physical RAM installed in the system.

Usage

```
get_total_ram()
```

Details

This function queries the operating system to determine total RAM. Works on Windows, Linux, and macOS.

The value returned is the physical RAM available to the system:

- On physical machines: actual installed RAM
- On virtual machines: RAM allocated to the VM
- On containers: RAM limit set for the container

Value

Numeric value with total RAM in gigabytes (GB)

See Also

`get_available_ram`, `get_cpu_cores`

Examples

```
# Check total RAM
total <- get_total_ram()
cat("System has", total, "GB of RAM\n")

# Returns 16.0 on a 16GB system
```

hdf5_apply *Apply a mathematical operation to multiple HDF5 datasets*

Description

Standalone function that applies a predefined algebraic or statistical operation to a list of datasets stored in an HDF5 group, writing results to `outgroup`. No open `HDF5Matrix` object is required.

This is not equivalent to `base::apply()`**.** It dispatches built-in C++ operations (QR, cross-product, Cholesky, etc.) to a batch of named datasets in the file.

Usage

```
hdf5_apply(
  filename,
  group,
  datasets,
  func,
  outgroup,
  b_group = NULL,
  b_datasets = NULL,
  overwrite = FALSE,
  transp_dataset = FALSE,
  transp_bdataset = FALSE,
  fullMatrix = FALSE,
  byrows = FALSE,
  threads = NULL
)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group path containing `datasets`.
`datasets`	Character vector of dataset names to process.
`func`	Character. Operation to apply (see Details).
`outgroup`	Character. Output group path for results.
`b_group`	Character or `NULL`. Group of B datasets.
`b_datasets`	Character vector or `NULL`. Names of B datasets.
`overwrite`	Logical. Overwrite existing output datasets.
`transp_dataset`	Logical. Transpose A datasets before operation.
`transp_bdataset`	Logical. Transpose B datasets before operation.
`fullMatrix`	Logical. Return full matrix (not triangular).
`byrows`	Logical. Apply by rows (for normalize/sdmean).
`threads`	Integer or `NULL`. OpenMP threads.

Details

Use `hdf5_apply()` when you only have the file path and group name. If you already have an open `HDF5Matrix`, use `apply_function(x, ...)` instead — both produce the same result.

`"CrossProd"` Compute $A^T A$ for each dataset.

`"tCrossProd"` Compute AA^T for each dataset.

`"CrossProd_double"` Double-precision cross-product.

`"tCrossProd_double"` Double-precision transposed cross-product.

`"blockmult"` Block-wise $A \times B$ (requires `b_datasets`).

`"QR"` QR decomposition for each dataset.

`"invChol"` Inverse via Cholesky for each dataset.

`"solve"` Solve $AX = B$ (requires `b_datasets`).

`"normalize"` Column-wise normalization.

`"sdmean"` Compute SD and mean.

`"descChol"` Cholesky decomposition.

Value

Invisibly `NULL`. Results written to `outgroup`. Open them with `hdf5_matrix()`.

See Also

`apply_function` for the S3 method on an open `HDF5Matrix`; `hdf5_reduce` for group-level reduction.

Examples

```
tmp <- tempfile(fileext = ".h5")
A <- hdf5_create_matrix(tmp, "inp/A", data = matrix(rnorm(25), 5, 5))
B <- hdf5_create_matrix(tmp, "inp/B", data = matrix(rnorm(25), 5, 5))
hdf5_apply(tmp, group = "inp", datasets = c("A", "B"),
func = "CrossProd", outgroup = "out")
res <- list_datasets(tmp)
res
res_A <- hdf5_matrix(tmp, res[3])
dim(res_A)   # 5 x 5
close(res_A)
hdf5_close_all()
unlink(tmp)
```

`hdf5_close_all` *Close all HDF5Matrix objects*

Description

Finds and closes all HDF5Matrix objects in the specified environment.

Usage

```
hdf5_close_all(envir = .GlobalEnv, verbose = TRUE)
```

Arguments

`envir`	Environment to search (default: .GlobalEnv)
`verbose`	Show details (default: TRUE)

Details

This function:

- Searches for HDF5Matrix objects in the environment
- Calls `$close()` on each valid object
- Forces garbage collection
- Reports closed files

Note: Only finds objects in the specified environment. Objects inside functions or other environments are not affected.

Value

Invisible vector of closed filenames

Examples

```
tmp1 <- tempfile(fileext = ".h5")
tmp2 <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp1, "data/A", data = matrix(rnorm(100), 10, 10))
Y  <- hdf5_create_matrix(tmp2, "data/B", data = matrix(rnorm(100), 10, 10))

X <- hdf5_matrix(tmp1, "data/A")
Y <- hdf5_matrix(tmp2, "data/B")

# Both open
X$is_valid()  # TRUE
Y$is_valid()  # TRUE

# Close all at once
```

```
hdf5_close_all()

# Both closed
X$is_valid()  # FALSE
Y$is_valid()  # FALSE

# Cleanup
unlink(c(tmp1, tmp2))
```

`hdf5_close_file` *Close all HDF5 handles for a specific file*

Description

Closes all open `HDF5Matrix` objects and HDF5 C library handles associated with a single HDF5 file, without affecting other open files.

Usage

```
hdf5_close_file(x)
```

Arguments

`x`	An `HDF5Matrix` object, or a character string with the path to the HDF5 file.

Value

Invisibly, the absolute path of the closed file.

See Also

`hdf5_close_all` to close all files at once.

Examples

```
fn1 <- tempfile(fileext = ".h5")
fn2 <- tempfile(fileext = ".h5")
A <- hdf5_create_matrix(fn1, "data/A", data = matrix(1:9, 3, 3))
B <- hdf5_create_matrix(fn2, "data/B", data = matrix(1:9, 3, 3))

# Close only fn1 — B remains open and usable
hdf5_close_file(fn1)
dim(B)   # still works

hdf5_close_all()
unlink(c(fn1, fn2))
```

`hdf5_create_matrix` *Create an HDF5 dataset and return an HDF5Matrix object*

Description

Creates a new HDF5 dataset (optionally writing data) and returns an `HDF5Matrix` object pointing to it.

Usage

```
hdf5_create_matrix(
  filename,
  dataset,
  nrow = NULL,
  ncol = NULL,
  data = NULL,
  dtype = c("double", "integer", "logical"),
  overwrite = FALSE,
  compression = NULL
)
```

Arguments

`filename`	Character. Path to the HDF5 file (created if it does not exist).
`dataset`	Character. Full path inside the HDF5 file in `"group/dataset"` or `"group/subgroup/dataset"` format.
`nrow`	Integer or NULL. Number of rows. Required when `data = NULL`.
`ncol`	Integer or NULL. Number of columns. Required when `data = NULL`.
`data`	Numeric matrix, integer matrix, or numeric vector, or NULL. When non-NULL, the data are written to the new dataset. When NULL, an empty (zero-filled) dataset of size `nrow x ncol` is created.
`dtype`	Character. Element type: `"double"` (default), `"integer"`, or `"logical"`.
`overwrite`	Logical. If `TRUE`, an existing dataset is replaced.
`compression`	Integer (0-9) or NULL. gzip compression level. `NULL` uses the global option set by `hdf5matrix_options` (default 6). Use `0` to disable compression.

Details

Replaces the legacy `bdCreate_hdf5_matrix()` / `bdCreate_hdf5_emptyDataset()` calls in the R6+S3 interface. The legacy functions remain available for backward compatibility.

Row and column names stored in the `dimnames` attribute of `data` are written to the HDF5 file automatically.

Value

An `HDF5Matrix` object pointing to the created dataset.

See Also

hdf5matrix_options to set global compression default.

Examples

```
tmp <- tempfile(fileext = ".h5")

# Create from matrix data
mat <- matrix(rnorm(200), nrow = 20, ncol = 10)
X <- hdf5_create_matrix(tmp, "data/X", data = mat)
dim(X)  # 20 x 10

# Create empty dataset
Y <- hdf5_create_matrix(tmp, "data/Y", nrow = 1000, ncol = 500)
dim(Y)  # 1000 x 500

# No compression (useful for benchmarks or intermediate results)
Z <- hdf5_create_matrix(tmp, "data/Z", data = mat, compression = 0)

X$close(); Y$close(); Z$close()
unlink(tmp)
```

---

hdf5_import *Import data from file or URL into HDF5 format*

Description

Modern wrapper for importing CSV, TSV, or other delimited text files into HDF5 format. Returns an HDF5Matrix object ready for use.

Usage

```
hdf5_import(
  source,
  filename,
  dataset,
  sep = NULL,
  header = TRUE,
  rownames = FALSE,
  overwrite = FALSE,
  parallel = TRUE,
  threads = NULL
)
```

Arguments

source	Character. Path to local file or URL to import. Supports compressed files (.gz, .tar.gz, .zip, .bz2).
filename	Character. Path to HDF5 output file (created if doesn't exist).
dataset	Character. Full dataset path (e.g., "data/imported" or "group/dataset").
sep	Character. Field separator. Default NULL (auto-detect from extension: "," for .csv, "\t" for .tsv, "\t" otherwise).
header	Logical or character vector. If TRUE, first row contains column names. If character vector, use these as column names. Default TRUE.
rownames	Logical or character vector. If TRUE, first column contains row names. If character vector, use these as row names. Default FALSE.
overwrite	Logical. If TRUE, overwrite dataset if exists. Default FALSE.
parallel	Logical. Use parallel processing for import. Default TRUE.
threads	Integer. Number of threads for parallel processing. Default NULL (uses all available cores).

Details

This function is a modern, user-friendly wrapper around bdImportData_hdf5 and bdImportTextFile_hdf5. It:

- Automatically detects file format from extension
- Handles compressed files (.gz, .tar.gz, .zip)
- Downloads from URLs automatically
- Returns ready-to-use HDF5Matrix object
- Uses sensible defaults for most use cases

Supported formats:

- CSV files (.csv) - comma-separated
- TSV files (.tsv, .txt) - tab-separated
- Compressed files (.gz, .tar.gz, .zip, .bz2)
- Remote files (http://, https://, ftp://)

Memory efficiency: Import is done in a streaming fashion, so very large files can be imported without loading them entirely into memory.

Value

HDF5Matrix object pointing to the imported data.

See Also

bdImportData_hdf5 for the underlying implementation, hdf5_create_matrix for creating matrices from R objects

Examples

```
csv_file  <- tempfile(fileext = ".csv")
hdf5_file <- tempfile(fileext = ".h5")

# Write sample numeric data
write.table(matrix(rnorm(50), nrow = 10, ncol = 5),
            csv_file, sep = ",", row.names = FALSE, col.names = TRUE)

# Import CSV to HDF5
mat <- hdf5_import(
  source   = csv_file,
  filename = hdf5_file,
  dataset  = "raw/data",
  sep      = ","
)
dim(mat)

hdf5_close_all()
unlink(c(csv_file, hdf5_file))
```

`hdf5_import_multiple` *Import multiple files into HDF5*

Description

Imports multiple files into the same HDF5 file, each as a separate dataset. Useful for batch importing related datasets.

Usage

```
hdf5_import_multiple(sources, filename, datasets, ...)
```

Arguments

`sources`	Character vector. Paths to files or URLs to import.
`filename`	Character. Path to HDF5 output file.
`datasets`	Character vector. Dataset paths for each source file. Must be same length as `sources`.
`...`	Additional arguments passed to `hdf5_import`

Value

Named list of `HDF5Matrix` objects, one for each imported file.

Examples

```
# Create temporary CSV files
f1 <- tempfile(fileext = ".csv")
f2 <- tempfile(fileext = ".csv")
f3 <- tempfile(fileext = ".csv")
hdf5_file <- tempfile(fileext = ".h5")

write.table(matrix(rnorm(20), 4, 5), f1, sep = ",",
            row.names = FALSE, col.names = TRUE)
write.table(matrix(rnorm(20), 4, 5), f2, sep = ",",
            row.names = FALSE, col.names = TRUE)
write.table(matrix(rnorm(20), 4, 5), f3, sep = ",",
            row.names = FALSE, col.names = TRUE)

mats <- hdf5_import_multiple(
  sources  = c(f1, f2, f3),
  filename = hdf5_file,
  datasets = c("data/exp1", "data/exp2", "data/exp3"),
  sep      = ","
)

dim(mats$exp1)

hdf5_close_all()
unlink(c(f1, f2, f3, hdf5_file))
```

---

hdf5_matrix *Open an HDF5 dataset as an HDF5Matrix object*

Description

Opens an existing dataset in an HDF5 file and returns an `HDF5Matrix` object that can be used with standard R syntax: `dim()`, `[i,j]`, `%*%`, `crossprod()`, and `tcrossprod()`.

Usage

```
hdf5_matrix(filename, path)
```

Arguments

`filename`	Path to an existing HDF5 file
`path`	Full path to the dataset within the file, in the form `"group/dataset"` or `"group/subgroup/dataset"`

Details

The HDF5 file remains open while the object exists, which improves performance for repeated operations. Call `$close()` to release the file lock explicitly, or use `rm()` and `gc()` for automatic cleanup. In an emergency, `hdf5_close_all` closes all open `HDF5Matrix` objects.

Value

An HDF5Matrix object

See Also

hdf5_close_all, [.HDF5Matrix, crossprod.HDF5Matrix, tcrossprod.HDF5Matrix

Examples

```
tmp <- tempfile(fileext = ".h5")

# Create dataset using BigDataStatMeth API
X <- hdf5_create_matrix(tmp, "data/expression",
                        data = matrix(rnorm(200), 20, 10))

dim(X)         # 20 x 10
X[1:5, 1:3]   # subset
crossprod(X)   # t(X) %*% X

close(X)
unlink(tmp)
```

---

hdf5_reduce *Reduce all datasets in an HDF5 group by a binary operation*

Description

Standalone function that applies a binary reduction ("+" or "-") across all datasets stored in a given HDF5 group and writes the accumulated result as a new dataset. No open HDF5Matrix object is required.

Usage

```
hdf5_reduce(
  filename,
  group,
  func = "+",
  outgroup = NULL,
  outdataset = NULL,
  overwrite = FALSE,
  remove = FALSE
)
```

Arguments

filename	Path to the HDF5 file.
group	Group path containing the datasets to reduce.
func	Character. Reduction operator: "+" (default) or "-".
outgroup	Character or NULL. Output group (default: same as group).
outdataset	Character or NULL. Output dataset name (default: same as group).
overwrite	Logical. Overwrite existing output dataset.
remove	Logical. Remove input datasets after reduction.

Details

Use hdf5_reduce() when you only have the file path and group name. If you already have an open HDF5Matrix, use reduce(x, ...) instead — both produce the same result.

All datasets in group must have the same dimensions.

Value

An HDF5Matrix pointing to the result dataset.

See Also

hdf5_apply, cbind.HDF5Matrix

Examples

```
fn <- tempfile(fileext = ".h5")
hdf5_create_matrix(fn, "blocks/A", data = matrix(1:6, 2, 3))
hdf5_create_matrix(fn, "blocks/B", data = matrix(1:6, 2, 3))
hdf5_create_matrix(fn, "blocks/C", data = matrix(1:6, 2, 3))
result <- hdf5_reduce(fn, group = "blocks", func = "+")
as.matrix(result)
hdf5_close_all()
unlink(fn)
```

---

hdf5_remove *Remove (delete) a dataset from an HDF5 file*

Description

Deletes a dataset from an HDF5 file, given the file path and the full internal dataset path ("group/name"). No open HDF5Matrix object is required. If you already have an open HDF5Matrix, you can equivalently call its $remove() method.

Usage

```
hdf5_remove(filename, dataset)
```

Arguments

filename	Path to the HDF5 file.
dataset	Full internal path of the dataset to remove, in the form "group/name" (e.g. "data/matrix" or "grp/sub/matrix").

Details

Space is not reclaimed. HDF5 only *unlinks* the dataset: the link is removed so the dataset can no longer be opened, but the disk space it occupied is **not** returned to the operating system until the file is rewritten. To physically reclaim the space, repack the file with the HDF5 command-line tool, e.g. `h5repack old.h5 new.h5`, or copy the datasets you want to keep into a fresh file. Consequently, repeatedly creating and removing datasets in the same file can make it grow on disk even though the logical contents shrink.

Removing a dataset that does not exist raises an error.

Value

`TRUE` invisibly on success.

See Also

hdf5_matrix, hdf5_create_matrix, list_datasets

Examples

```
fn <- tempfile(fileext = ".h5")
hdf5_create_matrix(fn, "grp/keep", data = matrix(1:6, 2, 3))
hdf5_create_matrix(fn, "grp/tmp",  data = matrix(1:6, 2, 3))
hdf5_remove(fn, "grp/tmp")
list_datasets(fn, group = "grp")   # only "keep" remains
hdf5_close_all()
unlink(fn)
```

HDF5Matrix-S3 *S3 methods for HDF5Matrix*

Description

Standard R generic methods for `HDF5Matrix` objects, allowing them to be used identically to in-memory matrices.

HDF5Matrix-scalar-aggregations

Summary statistics for HDF5Matrix

Description

Scalar aggregations over all elements of an HDF5 matrix, computed block-wise without loading the full data into RAM.

Usage

```
## S3 method for class 'HDF5Matrix'
mean(
  x,
  na.rm = FALSE,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)

## S3 method for class 'HDF5Matrix'
Summary(..., na.rm = FALSE)
```

Arguments

x	An HDF5Matrix object.
na.rm	Ignored (included for generic compatibility).
paral	Logical or NULL. Enable OpenMP parallelisation.
wsize	Integer or NULL. Block size (NULL = auto).
threads	Integer or NULL. Number of threads (NULL = auto).
save_to	Optional persistence target (same format as colSums.HDF5Matrix).
overwrite	Logical. Overwrite existing dataset when saving.
...	For Summary methods: additional objects (currently not supported; raises an error if supplied).

Value

A scalar numeric (when save_to = NULL) or an HDF5Matrix pointing to a 1×1 persisted dataset.

`hdf5matrix_options` *Set or get HDF5Matrix computation options*

Description

Configure global settings for parallelization, block processing and compression in HDF5Matrix operations. These settings affect all HDF5Matrix computations unless explicitly overridden in individual method calls.

Usage

```
hdf5matrix_options(
  paral = NULL,
  block_size = NULL,
  threads = NULL,
  compression = NULL
)
```

Arguments

`paral`	Logical or NULL. Enable OpenMP parallelization? • `TRUE`: Force parallel execution • `FALSE`: Force serial execution • `NULL`: Let BigDataStatMeth auto-detect (default)
`block_size`	Integer or NULL. Number of elements per block for block-wise processing. • Integer > 0: Use this block size • `NULL`: Auto-calculate based on matrix dimensions (default)
`threads`	Integer or NULL. Number of OpenMP threads to use. • Integer > 0: Use this many threads • `NULL`: Use all available threads (default)
`compression`	Integer (0-9) or NULL. gzip compression level for created datasets. • `0`: No compression (fastest, largest files) • `1-3`: Light compression (fast, moderate savings) • `6`: Balanced compression (default, 60-80\ • `7-9`: Maximum compression (slowest, best ratio) • `NULL`: Use built-in default of 6

Details

BigDataStatMeth achieves high performance through two key mechanisms:

Block-wise processing: Large matrices are processed in chunks that fit in memory. The `block_size` parameter controls chunk size. Smaller blocks use less memory but require more I/O operations. Larger blocks are faster but require more RAM.

OpenMP parallelization: Operations are distributed across CPU cores. The `paral` and `threads` parameters control this. Parallelization provides near-linear speedup for compute-intensive operations.

Compression: Datasets are created with gzip compression (level 6 by default). This reduces disk usage by 60-80\ For benchmarks or workflows where speed is critical, set `compression = 0`. For long-term storage or large datasets, keep the default.

Priority: Options set here serve as defaults. Individual method calls can override: `A$multiply(B, paral = TRUE, threads = 4, block_size = 2000)`

Recommendations:

- For interactive analysis: Leave defaults (`NULL`) - auto-detect works well
- For scripts/HPC: Set explicitly based on your hardware and data size
- For huge datasets (>10GB): Reduce `block_size` to fit in RAM
- For many-core systems: Set `threads` explicitly (auto may be too aggressive)
- For benchmarks: Set `compression = 0` to eliminate gzip overhead

Value

When called with arguments: invisibly returns a list of all current options. When called without arguments: returns a list of all current options.

Examples

```
# View current options
hdf5matrix_options()

# Enable parallelization with 8 threads
hdf5matrix_options(paral = TRUE, threads = 8)

# Set block size to 1000 elements
hdf5matrix_options(block_size = 1000)

# Disable compression for benchmarking
hdf5matrix_options(compression = 0)

# Reset to defaults
hdf5matrix_options(paral = NULL, threads = NULL, block_size = NULL, compression = NULL)
```

`impute_snps` *Impute missing SNP values in an HDF5Matrix*

Description

Fills the missing entries of SNP data by computing column or row means of the non-missing values. Intended for 0/1/2-coded diploid genotype matrices in which missing genotypes are stored as the value `3`; `NA` values must be recoded to `3` before calling this function.

Row and column names of `x`, when present, are carried over to the result.

Usage

```
impute_snps(x, ...)

## S3 method for class 'HDF5Matrix'
impute_snps(
  x,
  out_group = NULL,
  out_dataset = NULL,
  by_cols = TRUE,
  threads = -1L,
  overwrite = FALSE,
  ...
)
```

Arguments

`x`	An `HDF5Matrix` containing SNP data with missing values coded as `3`.
`...`	Ignored.
`out_group`	Output group. `NULL` = same as input (default).
`out_dataset`	Output dataset name. `NULL` = same as input (default, in-place).
`by_cols`	Logical. Impute by columns (`TRUE`, default) or rows.
`threads`	Integer. Number of threads (-1 = auto).
`overwrite`	Logical. Overwrite existing output. Default `FALSE`.

Value

`HDF5Matrix` pointing to the imputed dataset, carrying the dimension names of `x`.

Examples

```
set.seed(42)
tmp <- tempfile(fileext = ".h5")

# SNP data: 0/1/2 coded, 3 = missing (not NA)
snps <- matrix(sample(c(0L, 1L, 2L, 3L), 100 * 20,
                      replace = TRUE,
                      prob    = c(0.3, 0.3, 0.3, 0.1)),
               nrow = 100, ncol = 20)

X   <- hdf5_create_matrix(tmp, "geno/raw", data = snps)
imp <- impute_snps(X, out_group = "geno", out_dataset = "imputed")
```

```
dim(imp)

hdf5_close_all()
unlink(tmp)
```

`is_open` *Check if HDF5Matrix is open*

Description

Check whether an HDF5Matrix object is still valid and open.

Usage

```
is_open(x)
```

Arguments

`x`	An `HDF5Matrix` object

Value

Logical. `TRUE` if object is valid and open, `FALSE` otherwise.

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/matrix", data = matrix(rnorm(100), 10, 10))

X <- hdf5_matrix(tmp, "data/matrix")
is_open(X)  # TRUE

close(X)
is_open(X)  # FALSE

unlink(tmp)
```

length.HDF5Matrix *Length of an HDF5Matrix*

Description

Returns the total number of elements in an HDF5Matrix object, defined as prod(dim(x)) — consistent with the behaviour of base::length() for ordinary R matrices.

Usage

```
## S3 method for class 'HDF5Matrix'
length(x)
```

Arguments

x	An HDF5Matrix object.

Value

A single integer: nrow(x) * ncol(x).

Examples

```
tmp <- tempfile(fileext = ".h5")
X <- hdf5_create_matrix(tmp, "data/m", data = matrix(1:100, 10, 10))
length(X)   # 100
close(X); unlink(tmp)
```

list_datasets *List datasets in an HDF5 file or group*

Description

Lists datasets within an HDF5 file. If no group is specified, the entire file is traversed recursively and full relative paths are returned (e.g. "INPUT/A", "RESULTS/SVD/d"). If a group is given, only the datasets in that group are listed unless recursive = TRUE.

Usage

```
list_datasets(x, group = NULL, prefix = NULL, recursive = FALSE)
```

Arguments

x	An HDF5Matrix object, or a character string (file path).
group	Character or NULL. Group path inside the HDF5 file. If NULL and x is a file path, the whole file is listed recursively. If NULL and x is an HDF5Matrix, the object's own group is used (non-recursive).
prefix	Optional character. Only return datasets whose name starts with this prefix.
recursive	Logical. If TRUE, recurse into subgroups and return full relative paths. Default FALSE.

Value

Character vector of dataset names or relative paths.

See Also

hdf5_matrix, hdf5_create_matrix

Examples

```
fn <- tempfile(fileext = ".h5")
hdf5_create_matrix(fn, "INPUT/A",    data = matrix(rnorm(100), 10, 10))
hdf5_create_matrix(fn, "INPUT/B",    data = matrix(rnorm(100), 10, 10))
hdf5_create_matrix(fn, "RESULTS/C",  data = matrix(rnorm(100), 10, 10))

# All datasets in the file (recursive from root)
list_datasets(fn)

# Only INPUT group
list_datasets(fn, group = "INPUT")

# From an HDF5Matrix object (uses object's own group)
X <- hdf5_matrix(fn, "INPUT/A")
list_datasets(X)

hdf5_close_all()
unlink(fn)
```

---

memory_info *Print system memory information*

Description

Displays system memory information and current conversion thresholds. Useful for debugging and understanding memory limits.

Usage

```
memory_info()
```

Value

Invisible list with memory information

See Also

system_info, get_total_ram

Examples

```
memory_info()

# Conversion Thresholds:
#   Silent:  2.4 GB
#   Warning: 4.8 GB
#   Force:   8.0 GB
#   Blocked: 12.8 GB
```

---

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)
```

`multiply_sparse` *Sparse-aware matrix multiplication (generic)*

Description

Generic function for block-wise sparse matrix multiplication. The method for `HDF5Matrix` computes `x %*% y` using the BigDataStatMeth sparse multiplication algorithm, which skips all-zero blocks and is more efficient when one or both matrices are highly sparse.

Usage

```
multiply_sparse(x, y, ...)
```

Arguments

`x`	An `HDF5Matrix`.
`y`	An `HDF5Matrix`. Must be in the same HDF5 file as `x`.
`...`	Additional arguments forwarded to the method.

Value

A new `HDF5Matrix` containing the product.

See Also

`multiply_sparse.HDF5Matrix`

Examples

```
fn <- tempfile(fileext = ".h5")
X  <- hdf5_create_matrix(fn, "data/A", data = matrix(rnorm(100), 10, 10))
X  <- hdf5_create_matrix(fn, "data/B", data = matrix(rnorm(100), 10, 10))


A <- hdf5_matrix(fn, "data/A")
B <- hdf5_matrix(fn, "data/B")
C <- multiply_sparse(A, B)

hdf5_close_all()
unlink(fn)
```

`multiply_sparse.HDF5Matrix` *Sparse-aware matrix multiplication for HDF5Matrix*

Description

Computes `x %*% y` block-wise using BigDataStatMeth's sparse algorithm.

Usage

```
## S3 method for class 'HDF5Matrix'
multiply_sparse(
  x,
  y,
  outgroup = NULL,
  outdataset = NULL,
  block_size = -1L,
  mix_block = -1L,
  paral = NULL,
  threads = NULL,
  compression = NULL,
  ...
)
```

Arguments

`x`	An `HDF5Matrix`.
`y`	An `HDF5Matrix`. Same HDF5 file as `x`.
`outgroup`	Character or `NULL`. Output group in the HDF5 file. Default `"OUTPUT"`.
`outdataset`	Character or `NULL`. Output dataset name. Default is constructed from the operation and input names.
`block_size`	Integer. Block size hint; -1 = auto (default).
`mix_block`	Integer. Memory block size for parallel path; -1 = auto.
`paral`	Logical or NULL.
`threads`	Integer or NULL.
`compression`	Integer (0-9) or NULL.
`...`	Ignored.

Value

A new `HDF5Matrix`.

object_size *Get memory size of HDF5Matrix without loading*

Description

Estimates how much memory the dataset would occupy if loaded into RAM.

Usage

```
object_size(x, unit = c("MB", "bytes", "KB", "GB"))
```

Arguments

x	An HDF5Matrix object
unit	Character. Unit for size: "bytes", "KB", "MB", "GB". Default "MB".

Value

Numeric value with estimated memory size

Examples

```
fn <- tempfile(fileext = ".h5")
X <- hdf5_create_matrix(fn, "data/X", nrow = 100, ncol = 50)
object_size(X)
object_size(X, unit = "KB")
hdf5_close_all()
unlink(fn)
```

Ops.HDF5Matrix *Elementwise arithmetic operators for HDF5Matrix objects*

Description

Standard R arithmetic operators applied element-wise to HDF5Matrix objects stored on disk. Both operands must be HDF5Matrix objects with identical dimensions.

Usage

```
## S3 method for class 'HDF5Matrix'
Ops(e1, e2)
```

Arguments

`e1`	An `HDF5Matrix` object (left-hand side)
`e2`	An `HDF5Matrix` object (right-hand side)

Details

Supported operators:

- `+` Element-wise addition
- `-` Element-wise subtraction
- `*` Element-wise multiplication (Hadamard product)
- `/` Element-wise division. Division by zero produces `NaN` or `Inf`, matching base R behaviour.

All operations use block-wise processing and optional OpenMP parallelisation, controlled via `hdf5matrix_options`.

Broadcasting: if one operand is a $1 \times n$ or $n \times 1$ `HDF5Matrix` it is treated as a diagonal/vector operand and the operation is broadcast row- or column-wise (via `diag_op`) rather than applied element-wise. Both operands must be `HDF5Matrix` objects: scalar arithmetic such as `A + 5` is not supported.

Performance settings:

Global options set via `hdf5matrix_options` are applied. For explicit control use the R6 methods directly: `A$add(B, paral = TRUE, threads = 4)`.

Value

A new `HDF5Matrix` containing the result, stored in the same HDF5 file as `e1` under a temporary dataset name.

See Also

`hdf5matrix_options` for global performance settings, `HDF5Matrix` for R6 methods with explicit parameters

Examples

```
fn <- tempfile(fileext = ".h5")
A_hdf5 <- hdf5_create_matrix(fn, "data/A", data = matrix(1:12, 3, 4))
B_hdf5 <- hdf5_create_matrix(fn, "data/B", data = matrix(2, 3, 4))

C <- A_hdf5 + B_hdf5
D <- A_hdf5 - B_hdf5
E <- A_hdf5 * B_hdf5
G <- A_hdf5 / B_hdf5

all.equal(as.matrix(C), matrix(1:12, 3, 4) + 2)

hdf5_close_all()
unlink(fn)
```

`prcomp.HDF5Matrix` *Principal Component Analysis of an HDF5Matrix*

Description

Block-wise PCA entirely on disk, equivalent to `prcomp()`. Implements the same interface as `stats::prcomp()` but operates on data stored in an HDF5 file without loading it into RAM.

Usage

```
## S3 method for class 'HDF5Matrix'
prcomp(
  x,
  retx = TRUE,
  center = TRUE,
  scale. = FALSE,
  tol = NULL,
  rank. = NULL,
  ncomponents = 0L,
  k = 2L,
  q = 1L,
  method = "auto",
  rankthreshold = 0,
  svdgroup = "SVD/",
  overwrite = FALSE,
  threads = -1L,
  ...
)
```

Arguments

`x`	An `HDF5Matrix` object.
`retx`	Logical. If `TRUE` (default) return the individual coordinates (`x` slot). If `FALSE` the `x` slot is `NULL` in the returned object.
`center`	Logical. Subtract column means before PCA (default `TRUE`).
`scale.`	Logical. Divide by column SDs before PCA (default `FALSE`).
`tol`	Ignored (present for interface compatibility with `prcomp()`).
`rank.`	Integer. Number of principal components to compute. When supplied (non-`NULL`), takes precedence over `ncomponents`. Present for compatibility with `stats::prcomp()`; unlike the base R method, it is not ignored here.
`ncomponents`	Integer. Number of PCs to compute (0 = all, default).
`k`	Number of local SVDs per incremental level (default 2).
`q`	Number of incremental levels (default 1).
`method`	Computation method: `"auto"` (default), `"blocks"`, or `"full"`.

rankthreshold	Numeric in [0, 0.1]. Rank approximation threshold.
svdgroup	HDF5 group for intermediate SVD storage (default "SVD/").
overwrite	Logical. Recompute even if PCA results exist (default FALSE).
threads	Integer. OpenMP threads (-1 = auto-detect). The effective number may be lower than requested: the system ceiling (OMP_NUM_THREADS, OMP_THREAD_LIMIT and, by default, half of the detected cores) is applied silently by the OpenMP runtime. Requesting more than that is not an error, but it now emits a warning stating how many threads can actually be used.
...	Ignored (S3 compatibility).

Details

Constant columns and scale. = TRUE**.** As in stats::prcomp(), a column with zero variance cannot be rescaled to unit variance and the call stops with an error reporting how many columns are affected and a few of their positions. Previously the division by zero propagated silently and the whole decomposition degenerated to zeros. Either leave scale. = FALSE (the default) or remove the constant columns first.

Value

An object of class c("HDF5PCA", "list") with elements:

sdev Numeric vector. Standard deviations of the PCs.

rotation HDF5Matrix. Variable loadings (rotation matrix).

x HDF5Matrix or NULL. Individual coordinates.

center Logical. Whether columns were centered.

scale Logical. Whether columns were scaled.

cumvar Numeric vector. Cumulative variance explained (percent).

lambda Numeric vector. Eigenvalues.

var.cos2 HDF5Matrix. Squared cosines for variables.

ind.cos2 HDF5Matrix. Squared cosines for individuals.

ind.contrib HDF5Matrix. Contributions of individuals to PCs.

file Character. Path to the HDF5 file with all results.

The object also carries the attributes method, exact, elements, auto_threshold, blocking and rank, recording which decomposition path was actually taken; print() shows them.

Exact versus approximate decomposition

PCA runs on the same machinery as svd.HDF5Matrix and inherits both of its silent approximations. Read that help page before interpreting a large PCA. In particular, supplying ncomponents or rank. below min(dim(x)) switches on per-block rank truncation, which is the larger error source and can perturb even the first principal component by a few percent; asking for a generously oversampled number of components and discarding the extra ones is much more accurate than asking for exactly the few you want. Check attr(pca, "exact") and attr(pca, "truncated"); print() shows both.

Examples

```
tmp <- tempfile(fileext = ".h5")
X   <- hdf5_create_matrix(tmp, "data/M", data = matrix(rnorm(1000), 100, 10))
pca <- prcomp(X, center = TRUE, scale. = FALSE)
cat("Variance explained (PC1-3):", pca$cumvar[1:3], "\n")
dim(pca$rotation)   # 10 x nPC
dim(pca$x)          # 100 x nPC

# rank. takes precedence over ncomponents when both are supplied.
# NOTE: this reuses the same output location as the call above, so it
# must come after pca's results have already been read/used -- once
# overwrite = TRUE recreates the output datasets, any earlier
# HDF5Matrix handle pointing at that location (here, pca$rotation and
# pca$x) becomes invalid by design (see HDF5Matrix lifecycle, Section
# 5.3.3): the package proactively clears stale external pointers when
# overwriting to convert what would otherwise be a memory-unsafe crash
# into a clean R-level error.

pca5 <- prcomp(X, rank. = 5, ncomponents = 10, overwrite = TRUE)
dim(pca5$rotation)   # 10 x 5 -- rank. (5) was used, not ncomponents (10)

hdf5_close_all()
unlink(tmp)
```

---

print.HDF5Matrix *Print an HDF5Matrix object*

Description

Print an HDF5Matrix object

Usage

```
## S3 method for class 'HDF5Matrix'
print(x, ...)
```

Arguments

x	An HDF5Matrix object
...	Ignored

Value

Invisible x

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/matrix", data = matrix(rnorm(100), 10, 10))
X <- hdf5_matrix(tmp, "data/matrix")

print(X)
X   # same as print(X)

X$close()
unlink(tmp)
```

`print.HDF5PCA` *Print method for HDF5PCA objects*

Description

Print method for HDF5PCA objects

Usage

```
## S3 method for class 'HDF5PCA'
print(x, ...)
```

Arguments

`x`	An `HDF5PCA` object returned by `prcomp()` or `$pca()`.
`...`	Ignored.

`pseudoinverse` *Moore-Penrose pseudoinverse*

Description

Generic function for computing the Moore-Penrose pseudoinverse.

The `HDF5Matrix` method delegates to `bdpseudoinv_hdf5()`. This method reads the complete dataset into memory, computes the pseudoinverse via a direct LAPACK SVD, and writes the result back to HDF5. It is HDF5-backed but not block-wise: the full matrix and its pseudoinverse must both fit in available RAM during the computation. Result stored in the same HDF5 file under `OUTPUT/<dataset>_pinv` by default.

The `matrix` method delegates to `bdpseudoinv()` and computes the pseudoinverse of an in-memory matrix, returning a plain R matrix.

Both methods use the same singular-value tolerance (10^{-9}): singular values at or below the tolerance are treated as zero, following the standard Moore-Penrose construction $A^{+} = V\Sigma^{+}U^{\top}$.

Usage

```
pseudoinverse(x, ...)

## S3 method for class 'HDF5Matrix'
pseudoinverse(x, ...)

## S3 method for class 'matrix'
pseudoinverse(x, threads = NULL, ...)

## Default S3 method:
pseudoinverse(x, ...)
```

Arguments

x	An object: an HDF5Matrix, or a numeric matrix.
threads	Optional integer. Number of threads for parallel computation when x is a plain matrix. If NULL (default), uses the maximum available threads. Ignored for HDF5Matrix, where the thread count is passed via \dots instead (see below).
...	Additional arguments. For HDF5Matrix: passed to x$pseudoinverse(), i.e. outgroup, outdataset, overwrite, threads, compression.

Value

For HDF5Matrix: a new HDF5Matrix containing the pseudoinverse. For matrix: a plain numeric matrix.

See Also

solve.HDF5Matrix, svd.HDF5Matrix

Examples

```
## In-memory matrix
m <- matrix(c(1,2,3,4,5,6), 3, 2)
pseudoinverse(m)

## HDF5Matrix
tmp <- tempfile(fileext = ".h5")
X   <- hdf5_create_matrix(tmp, "data/A", data = m)
P   <- pseudoinverse(X)
dim(P)   # 2 x 3
close(X); close(P)
unlink(tmp)
```

qr *QR decomposition of an HDF5Matrix*

Description

Overrides `base::qr()` to dispatch on `HDF5Matrix` objects. For plain R matrices the call is forwarded to `base::qr()`.

Usage

```
qr(x, ...)
```

Arguments

`x`	An `HDF5Matrix` or a plain R matrix/vector.
`...`	Additional arguments passed to `qr.HDF5Matrix()` or ignored for `qr.default()`.

Value

For `HDF5Matrix`: a named list with elements `Q` and `R` (both `HDF5Matrix`). For plain R objects: the result of `base::qr()`.

See Also

`qr.HDF5Matrix`, `chol.HDF5Matrix`, `solve.HDF5Matrix`

qr.HDF5Matrix *QR decomposition of an HDF5Matrix*

Description

Computes A = Q R block-wise on disk and returns Q and R as `HDF5Matrix` objects.

Usage

```
## S3 method for class 'HDF5Matrix'
qr(
  x,
  thin = FALSE,
  block_size = NULL,
  overwrite = FALSE,
  threads = -1L,
  method = "auto",
  compression = NULL,
  ...
)
```

Arguments

x	An HDF5Matrix.
thin	Logical. Compute thin (economy) QR. Default FALSE. For method = "tsqr" this is strongly recommended; full Q via TSQR requires an expensive basis-completion step (O(m² k)).
block_size	Integer or NULL. Row-block size hint for TSQR; ignored by method = "lapack". NULL = auto.
overwrite	Logical. Overwrite existing results. Default FALSE.
threads	Integer. OpenMP threads (-1 = auto, CRAN-compliant). For method = "lapack" controls Eigen BLAS threads. For method = "tsqr" controls the OpenMP block loop.
method	Character. Algorithm selection: **"auto"** Default. Selects TSQR when m > 5n AND m > 1000 (tall-skinny), LAPACK otherwise. **"lapack"** Eigen HouseholderQR. Reliable for any matrix shape. **"tsqr"** Parallel Tall-Skinny QR. Requires m >= n. Efficient for big-omics tall matrices (e.g. samples × features). The R factor is mathematically equivalent to LAPACK R (up to sign). The Q factor is a valid orthogonal matrix but differs from LAPACK Q.
compression	Integer (0-9) or NULL. gzip compression level for the result datasets. NULL uses the global option set by hdf5matrix_options (default 1). Use 0 to disable compression (faster for benchmarks).
...	Ignored (for S3 compatibility).

Value

Named list: Q (HDF5Matrix), R (HDF5Matrix).

Note

20260304: Added method parameter and TSQR support.

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/A", data = matrix(rnorm(10000), 100, 100))

X   <- hdf5_matrix(tmp, "data/A")

# Default (auto method)
res <- qr(X)
dim(res$Q)   # m x m  (or m x min(m,n) if thin = TRUE)
dim(res$R)   # m x n  (or min(m,n) x n)

# Explicit TSQR for a tall-skinny matrix (recommended: thin = TRUE)
```

```
hdf5_close_all()
unlink(tmp)
```

`rbind.HDF5Matrix` *Row-bind HDF5Matrix objects*

Description

Binds two or more `HDF5Matrix` objects by rows (appending rows below). All matrices must have the same number of columns. The operation is performed block-wise on disk.

Usage

```
## S3 method for class 'HDF5Matrix'
rbind(
  ...,
  deparse.level = 1,
  out_file = NULL,
  out_group = NULL,
  out_dataset = NULL,
  block_rows = 1000L,
  overwrite = FALSE,
  compression = NULL
)
```

Arguments

`...`	One or more `HDF5Matrix` objects (all with the same number of columns).
`deparse.level`	Ignored (for S3 compatibility with base::rbind).
`out_file`	Output HDF5 file. `NULL` = same file as first argument.
`out_group`	Output group. `NULL` = `"BIND"`.
`out_dataset`	Output dataset name. `NULL` = auto-generated: for two inputs the name is `"A_rbind_B"`; for three or more inputs it is `"rbind_N"` where `N` is the number of inputs, to prevent unbounded name growth.
`block_rows`	Integer. Rows per I/O block (default 1000).
`overwrite`	Logical. Overwrite existing output. Default `FALSE`.
`compression`	Integer (0-9) or NULL. gzip compression level for the result datasets. NULL uses the global option set by `hdf5matrix_options` (default 1). Use `0` to disable compression (faster for benchmarks).

Value

`HDF5Matrix` pointing to the combined dataset.

Examples

```
fn <- tempfile(fileext = ".h5")

A  <- hdf5_create_matrix(fn, "grp/A", data = matrix(rnorm(100), 10, 10))
B  <- hdf5_create_matrix(fn, "grp/B", data = matrix(rnorm(100), 10, 10))

A <- hdf5_matrix(fn, "grp/A")
B <- hdf5_matrix(fn, "grp/B")
C <- rbind(A, B)          # rows of A followed by rows of B
dim(C)                    # (nrow(A) + nrow(B)) x ncol(A)

hdf5_close_all()
unlink(fn)
```

`reduce` *Reduce a group of HDF5 datasets by accumulation (generic)*

Description

Generic S3 function for reducing (accumulating) all datasets in the same HDF5 group as `x` into a single dataset using a binary operation (`"+"` or `"-"`).

This function operates on the **group** that contains `x`, not on the matrix data of `x` itself. All datasets in the group must have the same dimensions.

Usage

```
reduce(x, ...)
```

Arguments

`x`	An `HDF5Matrix`.
`...`	Additional arguments forwarded to the method.

Details

Two access patterns are available:

- `reduce(x, ...)` — use when you already have an `HDF5Matrix` object open. Operates on all datasets in its group.
- `hdf5_reduce(filename, group, ...)` — use when you only have the file path and group name, without needing to open an object.

This separation mirrors the pattern used by packages such as **DBI/RSQLite** (connection object vs. file path access).

Note: this is *not* a row- or column-wise reduction (use `colSums()`, `rowMeans()`, etc. for those). It accumulates entire datasets element-wise.

Value

A new HDF5Matrix containing the accumulated result.

See Also

hdf5_reduce for the standalone group-level version.

Examples

```
fn <- tempfile(fileext = ".h5")

# Create three matrices in the same group
hdf5_create_matrix(fn, "partials/chunk_0", data = matrix(1:100, 10, 10))
hdf5_create_matrix(fn, "partials/chunk_1", data = matrix(1:100, 10, 10))
hdf5_create_matrix(fn, "partials/chunk_2", data = matrix(1:100, 10, 10))

# Open one as entry point — reduce() operates on its whole group
partial <- hdf5_matrix(fn, "partials/chunk_0")
total   <- reduce(partial, func = "+")
dim(total)

hdf5_close_all()
unlink(fn)
```

---

scale *Scale / normalize an HDF5Matrix*

Description

Block-wise centering and scaling equivalent to base R scale(). The computation runs entirely on disk — the full matrix is never loaded into RAM.

Usage

```
## S3 method for class 'HDF5Matrix'
scale(
  x,
  center = TRUE,
  scale = TRUE,
  byrows = FALSE,
  wsize = NULL,
  result_path = NULL,
  compression = NULL,
  paral = NULL,
  threads = NULL,
  ...
)
```

Arguments

x	An HDF5Matrix object.
center	Logical (or numeric vector, see Details). If TRUE (default) subtract column means before scaling.
scale	Logical (or numeric vector, see Details). If TRUE (default) divide by column standard deviations.
byrows	Logical. If TRUE normalize row-wise instead of column-wise. Default FALSE.
wsize	Integer or NULL. Block size for HDF5 reads (NULL = auto).
result_path	Output location. NULL (default) writes to "NORMALIZED/<group>/<dataset>" in the same file. A character string writes to that path in the same file. A named list list(file=, path=) writes to a different file.
compression	Integer (0-9) or NULL. gzip compression level for the result datasets. NULL uses the global option set by hdf5matrix_options (default 1). Use 0 to disable compression (faster for benchmarks).
paral	Logical or NULL. Enable OpenMP parallelism. TRUE forces block-wise streaming (PATH 2) regardless of matrix size, so the thread count is respected. NULL or FALSE uses the preload path (PATH 1) when the matrix fits in RAM. Overrides the global option set by hdf5matrix_options. Default NULL.
threads	Integer or NULL. Number of OpenMP threads when paral = TRUE. Always capped by OMP_THREAD_LIMIT (CRAN compliance). NULL uses the system default. Overrides the global option set by hdf5matrix_options.
...	Ignored (for S3 compatibility).

Details

Passing a pre-computed numeric vector as center or scale is not currently supported. If a vector is supplied it is coerced to a logical (TRUE if length(x) > 0) and a warning is issued.

The returned HDF5Matrix carries scaled:center and scaled:scale attributes (numeric vectors), mirroring the behavior of base::scale().

Performance settings:

Parallelization and thread count can be set globally via hdf5matrix_options or passed explicitly via paral and threads. Explicit parameters take priority over global options.

```
# Global configuration
hdf5matrix_options(paral = TRUE, threads = 4)
Xs <- scale(X)           # uses 4 threads

# Explicit per-call override
Xs <- scale(X, paral = TRUE, threads = 8)
```

Value

An HDF5Matrix pointing to the normalized dataset on disk.

See Also

hdf5matrix_options for global performance settings.

Examples

```
tmp <- tempfile(fileext = ".h5")
X   <- hdf5_create_matrix(tmp, "data/M",
                          data = matrix(rnorm(500), 50, 10))
Xs  <- scale(X)                          # center=TRUE, scale=TRUE by cols
cat("scaled:center[1]:", attr(Xs, "scaled:center")[1], "\n")
X$close(); Xs$close(); unlink(tmp)
```

---

sd *Standard deviation of all elements of an HDF5Matrix*

Description

Equivalent to `sd(as.vector(X))` — uses Bessel's correction (N-1).

Usage

```
sd(x, na.rm = FALSE, ...)

## S3 method for class 'HDF5Matrix'
sd(
  x,
  na.rm = FALSE,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

x	An HDF5Matrix object.
na.rm	Ignored (included for generic compatibility).
...	For Summary methods: additional objects (currently not supported; raises an error if supplied).
paral	Logical or NULL. Enable OpenMP parallelisation.
wsize	Integer or NULL. Block size (NULL = auto).
threads	Integer or NULL. Number of threads (NULL = auto).
save_to	Optional persistence target (same format as colSums.HDF5Matrix).
overwrite	Logical. Overwrite existing dataset when saving.

Value

Scalar numeric or an HDF5Matrix when save_to is set.

---

show_hdf5matrix_options *Show current HDF5Matrix performance settings*

Description

Display current global options in a user-friendly format.

Usage

```
show_hdf5matrix_options()
```

Value

Invisibly returns the options list

Examples

```
show_hdf5matrix_options()
```

---

solve.HDF5Matrix *Matrix inverse of a symmetric positive-definite HDF5Matrix via Cholesky*

Description

Computes the matrix inverse of a symmetric positive-definite HDF5Matrix using Cholesky decomposition + back-substitution. Equivalent to base::solve(A) for SPD matrices.

Usage

```
## S3 method for class 'HDF5Matrix'
solve(
  a,
  b,
  full_matrix = TRUE,
  overwrite = FALSE,
  threads = -1L,
  block_size = NULL,
  compression = NULL,
  ...
)
```

Arguments

a	An HDF5Matrix (square, symmetric positive-definite).
b	Not supported for HDF5Matrix; must be missing.
full_matrix	Logical. Return full symmetric inverse. Default TRUE.
overwrite	Logical. Overwrite existing result. Default FALSE.
threads	Integer. OpenMP threads (-1 = auto).
block_size	Integer or NULL. Elements per block. NULL = auto.
compression	Integer (0-9) or NULL. gzip compression level for the result dataset. NULL uses the global option set by hdf5matrix_options (default 1). Use 0 to disable compression (faster for benchmarks).
...	Ignored (for S3 compatibility).

Value

HDF5Matrix containing the matrix inverse.

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/X", data = matrix(rnorm(10000), 100, 100))

X   <- hdf5_matrix(tmp, "data/X")
AtA <- crossprod(X)              # HDF5Matrix, square SPD
inv <- solve(AtA)               # inverse of AtA

hdf5_close_all()
unlink(tmp)
```

---

split.HDF5Matrix *Split an HDF5Matrix into a list of blocks*

Description

S3 method of base::split() for HDF5Matrix objects. Divides the matrix into equal-sized blocks along rows (default) or columns, storing each block as a separate dataset in the same HDF5 file.

Provide exactly ONE of n_blocks or block_size.

Usage

```
## S3 method for class 'HDF5Matrix'
split(
  x,
  f = NULL,
  drop = FALSE,
  n_blocks = -1L,
  block_size = -1L,
  bycols = FALSE,
  out_group = "SPLIT",
  out_dataset = NULL,
  overwrite = FALSE,
  ...
)
```

Arguments

x	An HDF5Matrix.
f	Ignored (kept for S3 signature compatibility).
drop	Ignored (S3 compatibility).
n_blocks	Integer. Number of (roughly equal) blocks; -1 = unused.
block_size	Integer. Max rows (or cols) per block; -1 = unused.
bycols	Logical. If TRUE, split by columns (default = by rows).
out_group	Character. HDF5 group for output blocks (default "SPLIT").
out_dataset	Character or NULL. Base dataset name.
overwrite	Logical. Overwrite existing blocks (default FALSE).
...	Ignored.

Details

Calling convention: use `split(x, n_blocks = 4)` after loading the package. The form `BigDataStatMeth::split()` produces an error because `split` is a `base` generic — this is normal R behaviour, identical to `BigDataStatMeth::cor()` or `BigDataStatMeth::svd()`. The S3 dispatch happens automatically when the package is loaded.

`split_dataset()` is an alternative with a cleaner signature that omits the `f` and `drop` parameters inherited from `base::split` (which have no meaning for HDF5Matrix objects). Both functions produce identical results.

Value

Named list of `HDF5Matrix` objects: `block_0`, `block_1`, ...

See Also

`split_dataset` for the equivalent with a cleaner signature; `hdf5_reduce` to recombine blocks after processing.

Examples

```
fn     <- tempfile(fileext = ".h5")
X      <- hdf5_create_matrix(fn, "data/X", data = matrix(rnorm(2000), 20, 100))
blocks <- split(X, n_blocks = 4)   # 4 row-blocks of 5 rows each
length(blocks)                     # 4
lapply(blocks, close)

hdf5_close_all()
unlink(fn)
```

`split_dataset` *Split an HDF5Matrix into multiple block datasets*

Description

Splits an `HDF5Matrix` into equal-sized sub-matrices stored as separate datasets in the same HDF5 file. This is the preferred form when you want an explicit, unambiguous call: unlike `split()`, it does not carry the `f` and `drop` parameters inherited from `base::split` that have no meaning for HDF5Matrix objects.

Exactly one of `n_blocks` or `block_size` must be provided. Output datasets are named `<out_group>/<out_dataset>.0`, `<out_group>/<out_dataset>.1`, ... (0-based index).

Usage

```
split_dataset(x, n_blocks = NULL, block_size = NULL, bycols = FALSE, ...)

## S3 method for class 'HDF5Matrix'
split_dataset(
  x,
  n_blocks = NULL,
  block_size = NULL,
  bycols = FALSE,
  out_group = "SPLIT",
  out_dataset = NULL,
  overwrite = FALSE,
  ...
)
```

Arguments

`x`	An `HDF5Matrix`.
`n_blocks`	Integer or `NULL`. Number of blocks.
`block_size`	Integer or `NULL`. Rows or columns per block.
`bycols`	Logical. Split by columns (`TRUE`) or rows (default `FALSE`).

`...`	Ignored.
`out_group`	Character. Output HDF5 group (default `"SPLIT"`).
`out_dataset`	Character or NULL. Base dataset name.
`overwrite`	Logical. Overwrite existing blocks (default `FALSE`).

Value

A named list of `HDF5Matrix` objects.

See Also

`split.HDF5Matrix` for the `base::split()` S3 dispatch equivalent; `hdf5_reduce` to recombine blocks.

Examples

```
tmp  <- tempfile(fileext = ".h5")
M    <- hdf5_create_matrix(tmp, "data/M", data = matrix(1:60, 6, 10))
blks <- split_dataset(M, n_blocks = 3L)
length(blks)
lapply(blks, close)
close(M)
unlink(tmp)
```

`str.HDF5Matrix` *Structure of an HDF5Matrix object*

Description

Structure of an HDF5Matrix object

Usage

```
## S3 method for class 'HDF5Matrix'
str(object, ...)
```

Arguments

`object`	An `HDF5Matrix` object
`...`	Ignored

Value

Invisible `object`

Examples

```
tmp <- tempfile(fileext = ".h5")

X  <- hdf5_create_matrix(tmp, "data/matrix", data = matrix(rnorm(100), 10, 10))
X <- hdf5_matrix(tmp, "data/matrix")

str(X)

X$close()
unlink(tmp)
```

---

svd *Singular Value Decomposition (generic)*

Description

S3 generic for `svd()`. Dispatches to `svd.HDF5Matrix` for `HDF5Matrix` objects, and to `base::svd()` for all others.

Usage

```
svd(x, nu = min(dim(x)), nv = min(dim(x)), ...)
```

Arguments

`x`	A matrix or `HDF5Matrix` object.
`nu`	Number of left singular vectors to compute.
`nv`	Number of right singular vectors to compute.
`...`	Additional arguments passed to the method.

Value

Named list with components `d`, `u`, `v`.

svd.HDF5Matrix *Singular Value Decomposition of an HDF5Matrix*

Description

Block-wise SVD entirely on disk. The matrix x is decomposed into x = u %*% diag(d) %*% t(v).

Usage

```
## S3 method for class 'HDF5Matrix'
svd(
  x,
  nu = min(dim(x)),
  nv = min(dim(x)),
  center = TRUE,
  scale = TRUE,
  k = 2L,
  q = 1L,
  method = "auto",
  rankthreshold = 0,
  overwrite = FALSE,
  threads = -1L,
  ...
)
```

Arguments

x	An HDF5Matrix object.
nu	Number of left singular vectors to compute (default = min(dim(x))).
nv	Number of right singular vectors to compute (default = min(dim(x))).
center	Logical. Center columns before decomposition (default TRUE).
scale	Logical. Scale columns before decomposition (default TRUE).
k	Number of local SVDs per incremental level (default 2).
q	Number of incremental levels (default 1).
method	Computation method: "auto" (default), "blocks", or "full".
rankthreshold	Numeric in [0, 0.1]. Rank approximation threshold (default 0).
overwrite	Logical. Overwrite existing SVD results (default FALSE).
threads	Integer. OpenMP threads (-1 = auto-detect). The effective number may be lower than requested: the system ceiling (OMP_NUM_THREADS, OMP_THREAD_LIMIT and, by default, half of the detected cores) is applied silently by the OpenMP runtime. Requesting more than that is not an error, but it now emits a warning stating how many threads can actually be used.
...	Ignored (S3 compatibility).

Details

Singular values `d` are loaded into a plain numeric vector (they are always small: at most `min(nrow(x), ncol(x))` values). `u` and `v` are returned as `HDF5Matrix` objects.

Constant columns and `scale = TRUE`. A column with zero variance cannot be rescaled to unit variance, so the call stops with an error reporting how many columns are affected and a few of their positions – the same contract as `stats::prcomp(x, scale. = TRUE)`. Previously the division by zero propagated silently and every singular value came back as `0`. Either pass `scale = FALSE` or remove the constant columns first. `scale.HDF5Matrix` is deliberately unaffected: like `base::scale()` it still returns `NaN` for such a column.

Value

Named list with:

`d` Numeric vector of non-negative singular values, decreasing.

`u` HDF5Matrix of left singular vectors, `nrow(x)` x `nu`.

`v` HDF5Matrix of right singular vectors, `ncol(x)` x `nv`.

The list carries attributes describing what was actually computed:

`method` `"full"` (exact LAPACK) or `"blocks"` (hierarchical block algorithm).

`exact` `TRUE` when the exact LAPACK algorithm was used, i.e. `method == "full"` and no per-block truncation. It reports the *algorithm*, not a measured accuracy: the block path at full rank is not exact by construction, though in our tests it agreed with LAPACK to ~1e-15 (see below).

`nev` Per-block truncation rank actually applied (`0` = none). Requesting fewer vectors below the boundary does not truncate anything: the exact path runs and the reduction happens afterwards in R, losslessly.

`truncated` `TRUE` when per-block truncation was applied, which is the dominant source of error (see below).

`elements` `nrow(x) * ncol(x)`, the quantity compared against the boundary.

`auto_threshold` The boundary itself, see `svd_auto_threshold`.

`blocking` The `k` and `q` actually used.

`rank` Number of singular values returned.

Exact versus approximate decomposition

This is the most important behavioural difference from `base::svd()`. **Two independent things** can make the result approximate, and they are not equally dangerous.

1. The algorithm (minor). `method = "full"` reads the whole matrix into RAM and computes an exact LAPACK SVD, needing `nrow * ncol * 8` bytes. `method = "blocks"` computes a hierarchical block SVD entirely on disk. `method = "auto"` (the default) picks `"full"` while `nrow * ncol < svd_auto_threshold()` (currently 53,687,091 elements, about 410 MB of doubles) and `"blocks"` at or above it. When the full rank is requested, the block path is essentially exact: relative errors of order 1e-15 across the whole spectrum, independent of `k` and `q`, in our measurements. Crossing the boundary is, on its own, not a numerical event.

2. Per-block rank truncation (major). Asking for fewer singular triplets than `min(dim(x))` – `nu` or `nv` below full rank, or `ncomponents`/`rank.` in `prcomp.HDF5Matrix` – switches on truncation of every *local* block SVD to that rank before the blocks are merged. That is what actually costs accuracy, and it compounds with blocking depth. On a 1600 x 200 matrix with a decaying spectrum, relative error in the singular values:

requested rank	k, q	leading	trailing
200 (full)	2, 1	3e-16	7e-15
200 (full)	4, 3	1e-15	4e-15
50	2, 1	7e-06	8e-02
50	2, 3	5e-05	1e-01
10	2, 1	2e-03	1e-01
10	2, 3	1e-02	2e-01

Note that the error is not confined to the tail: at rank 10 even the *leading* singular value was wrong by 1-3\ of accurate components, ask for a generously oversampled rank and discard the extra ones – in the table above, computing 50 and keeping 10 is about 250 times more accurate in the leading value than computing 10 directly – or request the full rank when you can afford it.

Detecting it. Both switches are silent by default, so the result records them: check `attr(res, "exact")`, `attr(res, "truncated")` and `attr(res, "method")`. `svd()` also emits a `message()` when `"auto"` falls through to the block path and when per-block truncation is applied (silence with `suppressMessages()`).

There is no cheap residual diagnostic: verifying a decomposition means reconstructing `u %*% diag(d) %*% t(v)` and comparing it with `x`, another full pass over the data. The practical checks are to recompute with a larger requested rank, or with `method = "full"` on a subset, and see whether the leading components move.

There is no cheap residual diagnostic: verifying the decomposition means reconstructing `u %*% diag(d) %*% t(v)` and comparing it with `x`, which costs another full pass over the data. When accuracy matters, the practical check is to recompute a small trailing portion with `method = "full"` on a subset, or to compare across two values of `q`.

Examples

```
tmp <- tempfile(fileext = ".h5")
X   <- hdf5_create_matrix(tmp, "data/M", data = matrix(rnorm(500), 50, 10))
res <- svd(X)
length(res$d)   # 10  (min(50,10))
dim(res$u)      # 50 x 10
dim(res$v)      # 10 x 10
X$close()
res$u$close(); res$v$close()
unlink(tmp)
```

svd_auto_threshold *Element-count boundary of the automatic SVD/PCA path selection*

Description

Number of matrix elements (`nrow * ncol`) at or above which `method = "auto"` stops computing an exact decomposition and switches to the hierarchical block approximation.

Usage

```
svd_auto_threshold()
```

Details

The value is read from the C++ constant `SVD_EXACT_MAX_ELEMENTS` so that documentation and implementation cannot drift apart. It currently corresponds to 53,687,091 elements, about 410 MB of double-precision data.

Value

Numeric scalar: the element-count boundary.

See Also

`svd.HDF5Matrix`, `prcomp.HDF5Matrix`

Examples

```
svd_auto_threshold()
```

sweep *Sweep out array summaries (generic)*

Description

S3 generic for `sweep()`. Dispatches to `sweep.HDF5Matrix` for `HDF5Matrix` objects, and to `base::sweep()` for all others.

Usage

```
sweep(x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)
```

Arguments

x	A matrix or HDF5Matrix object.
MARGIN	Integer. 1 for rows, 2 for columns.
STATS	Numeric vector to sweep out.
FUN	Character. Function to apply: "-", "+", etc.
check.margin	Logical. Check that STATS length matches margin.
...	Additional arguments passed to the method.

Value

A new HDF5Matrix or matrix with STATS swept out.

---

sweep.HDF5Matrix *Broadcast a vector over an HDF5Matrix (sweep)*

Description

S3 method of base::sweep() for HDF5Matrix objects. Broadcasts a 1-row HDF5Matrix (acting as the STATS vector) across every row or column of the matrix, element-wise.

Usage

```
## S3 method for class 'HDF5Matrix'
sweep(
  x,
  MARGIN = 2L,
  STATS,
  FUN = "*",
  check.margin = TRUE,
  paral = NULL,
  threads = NULL,
  compression = NULL,
  ...
)
```

Arguments

x	An HDF5Matrix (the matrix to sweep over).
MARGIN	Integer. 2 (default, sweep over columns) or 1 (sweep over rows). Corresponds to byrows: MARGIN = 1 means byrows = TRUE.
STATS	An HDF5Matrix with one row or one column (the vector to broadcast). Must be in the same HDF5 file as x.
FUN	Character. Operation: "+", "-", "*" (default), "/", "pow".
check.margin	Ignored (kept for S3 signature compatibility).

paral	Logical or NULL.
threads	Integer or NULL.
compression	Integer (0-9) or NULL.
...	Ignored.

Value

A new HDF5Matrix.

Examples

```
fn <- tempfile(fileext = ".h5")

mat <- matrix(rnorm(100), 10, 10)
X   <- hdf5_create_matrix(fn, "data/X", data = mat)

# STATS must be an HDF5Matrix with one row or one column
# Create a 1-row vector with column means
col_means_vec <- colMeans(as.matrix(X))
stats_hdf5    <- hdf5_create_matrix(fn, "data/col_means",
                                    data = matrix(col_means_vec, 1, 10))

# Column-center X (MARGIN = 2)
X_c <- sweep(X, 2, stats_hdf5, "-")

# Verify first column is centered
all.equal(as.matrix(X_c)[, 1],
          mat[, 1] - col_means_vec[1])

hdf5_close_all()
unlink(fn)
```

---

system_info *Get system information summary*

Description

Returns a comprehensive summary of system resources.

Usage

```
system_info()
```

Details

Convenience function that calls all system info methods and returns a summary. Useful for debugging and logging.

Value

Named list with system information:

os Operating system name

total_ram_gb Total RAM in GB

available_ram_gb Available RAM in GB

ram_used_pct Percentage of RAM currently used

cpu_cores Number of CPU cores

Examples

```
# Get full system info
info <- system_info()
print(info)
```

`tcrossprod` *Transposed cross product of HDF5Matrix objects*

Description

S3 generic for `tcrossprod()`. Dispatches to `tcrossprod.HDF5Matrix` for `HDF5Matrix` objects, and to `base::tcrossprod()` for all others.

Usage

```
tcrossprod(x, y = NULL, ...)

## S3 method for class 'HDF5Matrix'
tcrossprod(x, y = NULL, outgroup = NULL, outdataset = NULL, ...)
```

Arguments

`x`	An `HDF5Matrix` object.
`y`	An `HDF5Matrix` object, or `NULL` (default) to compute `x %*% t(x)`.
`...`	Ignored.
`outgroup`	Character or `NULL`. HDF5 group where the result is stored. Default `"OUTPUT"`.
`outdataset`	Character or `NULL`. Dataset name for the result. Default `"tCrossProd_x"` (single matrix) or `"tCrossProd_x_x_y"` (two matrices).

Details

Computes $x \times t(y)$ (or $x \times t(x)$ when `y = NULL`). Uses the dedicated BigDataStatMeth block-wise transposed cross-product algorithm, which is more efficient than explicitly computing `x %*% t(y)`.

Performance settings:

This method uses global options set via `hdf5matrix_options`.

Symmetric optimization:

When `y = NULL` or `y` refers to the same dataset as `x`, the symmetric optimisation is applied automatically, providing significant speedup.

Value

Result of the cross product.

A new `HDF5Matrix` pointing to the result dataset.

See Also

`hdf5matrix_options` for global performance settings

Examples

```
fn <- tempfile(fileext = ".h5")
X <- hdf5_create_matrix(fn, "INPUT/X", data = matrix(rnorm(60), 6, 10))
Y <- hdf5_create_matrix(fn, "INPUT/Y", data = matrix(rnorm(60), 6, 10))

# t(X) %*% X  → stored in OUTPUT/CrossProd_X
C1 <- tcrossprod(X)
dim(C1)

# t(X) %*% Y  → stored in OUTPUT/CrossProd_X_x_Y
C2 <- tcrossprod(X, Y)

# Custom output location
C3 <- tcrossprod(X, outgroup = "RESULTS", outdataset = "my_tcrossprod")

hdf5_close_all()
unlink(fn)
```

---

var *Variance of all elements of an HDF5Matrix*

Description

Equivalent to `var(as.vector(X))` — treats all matrix elements as a single sample and uses Bessel's correction (N-1).

Usage

```
var(x, y = NULL, na.rm = FALSE, use, ...)

## S3 method for class 'HDF5Matrix'
var(
  x,
  y = NULL,
  na.rm = FALSE,
  use,
  paral = NULL,
  wsize = NULL,
  threads = NULL,
  save_to = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

x	An HDF5Matrix object.
y	Ignored. Present for compatibility with stats::var.
na.rm	Ignored (included for generic compatibility).
use	Ignored. Present for compatibility with stats::var.
...	For Summary methods: additional objects (currently not supported; raises an error if supplied).
paral	Logical or NULL. Enable OpenMP parallelisation.
wsize	Integer or NULL. Block size (NULL = auto).
threads	Integer or NULL. Number of threads (NULL = auto).
save_to	Optional persistence target (same format as colSums.HDF5Matrix).
overwrite	Logical. Overwrite existing dataset when saving.

Value

Scalar numeric or an HDF5Matrix when save_to is set.

Index

∗ **BigDataStatMeth HDF5 utilities**
    bdmove_hdf5_dataset, 30
∗ **datasets**
    cancer, 44
    miRNA, 95
[.HDF5Matrix, 4, *10*, *85*
[<-.HDF5Matrix, 5
%*%, 6

apply_function, 7, *77*
as.data.frame.HDF5Matrix, 8, *10*
as.matrix.HDF5Matrix, 9, *9*

bd_wproduct, 11
bdapply_Function_hdf5, 12
bdblockMult, 14, *17*, *19*, *24*, *39*
bdblockSubstract, *15*, 16, *19*
bdblockSum, *15*, *17*, 18
bdCorr_matrix, 20
bdCreate_hdf5_group, 21
bdCreate_hdf5_matrix, 22, *35*
bdCrossprod, 23, *39*
bdgetDatasetsList_hdf5, 25
bdImportData_hdf5, 26, *82*
bdImportTextFile_hdf5, 28, *82*
bdmove_hdf5_dataset, 30
bdpseudoinv, 32, *35*
bdpseudoinv_hdf5, 34
bdReduce_hdf5_dataset, 36
bdScalarwproduct, 37
bdtCrossprod, *24*, 38
bdWrite_hdf5_dimnames, 40
BigDataStatMeth, 41

can_allocate, 43, *71*
cancer, 44
cbind.HDF5Matrix, 45, *86*
chol.HDF5Matrix, 46, *105*
close.HDF5Matrix, 47
colesterol, 48

colMaxs, 49
colMeans, 50
colMins, 51
colnames.HDF5Matrix
        *(*dimnames.HDF5Matrix*)*, 64
colnames<-.HDF5Matrix
        *(*dimnames.HDF5Matrix*)*, 64
colSds, 53
colSums, 54
colSums.HDF5Matrix, *88*, *111*, *126*
colVars, 55
cor, 56
cor.HDF5Matrix, *56*, 57
crossprod, 59
crossprod.HDF5Matrix, *8*, *59*, *85*

diag, 60, *61*
diag<-, 63
diag_op, 61, *63*, *99*
diag_scale, 62, *62*
dim.HDF5Matrix, 64
dimnames.HDF5Matrix, 64
dimnames<-.HDF5Matrix, 66

eigen, 66

filter_low_coverage, 67
filter_maf, 69

get_available_ram, *44*, 70, *75*
get_cpu_cores, 71, *74, 75*
get_memory_thresholds, 72
get_recommended_threads, 74
get_total_ram, *71–73*, 75, *95*

hdf5_apply, *7, 8*, 76, *86*
hdf5_close_all, *48*, 78, *79*, *84, 85*
hdf5_close_file, 79
hdf5_create_matrix, *22*, *45*, 80, *82*, *87*, *94*
hdf5_import, 81, *83*
hdf5_import_multiple, 83

hdf5_matrix, *45*, *48*, *77*, 84, *87*, *94*
hdf5_reduce, *77*, 85, *108*, *109*, *114*, *116*
hdf5_remove, 86
HDF5Matrix-S3, 87
HDF5Matrix-scalar-aggregations, 88
hdf5matrix_options, *45*, *47*, *58–60*, *80*, *81*, 89, *99*, *106*, *107*, *110*, *111*, *113*, *125*

impute_snps, 90
is_open, 92

length.HDF5Matrix, 93
list_datasets, *87*, 93

mean.HDF5Matrix *(*HDF5Matrix-scalar-aggregations*)*, 88
memory_info, *73*, 94
miRNA, 95
multiply_sparse, 96
multiply_sparse.HDF5Matrix, *96*, 97

object_size, 98
Ops.HDF5Matrix, 98

prcomp, *100*
prcomp.HDF5Matrix, *67*, 100, *120*, *121*
print.HDF5Matrix, 102
print.HDF5PCA, 103
pseudoinverse, 103

qr, 105
qr.HDF5Matrix, *8*, 105, *105*

rbind.HDF5Matrix, 107
reduce, *86*, 108
rowMaxs *(*colMaxs*)*, 49
rowMeans *(*colMeans*)*, 50
rowMins *(*colMins*)*, 51
rownames.HDF5Matrix *(*dimnames.HDF5Matrix*)*, 64
rownames<-.HDF5Matrix *(*dimnames.HDF5Matrix*)*, 64
rowSds *(*colSds*)*, 53
rowSums *(*colSums*)*, 54
rowVars *(*colVars*)*, 55

scale, 109, *109*
scale.HDF5Matrix, *119*
sd, 111
show_hdf5matrix_options, 112
solve.HDF5Matrix, *104*, *105*, 112
split.HDF5Matrix, 113, *116*
split_dataset, *114*, 115
str.HDF5Matrix, 116
Summary.HDF5Matrix *(*HDF5Matrix-scalar-aggregations*)*, 88
svd, 117
svd.HDF5Matrix, *67*, *101*, *104*, *117*, 118, *121*
svd_auto_threshold, *119*, 121
sweep, 121
sweep.HDF5Matrix, *121*, 122
system_info, *95*, 123

tcrossprod, 124
tcrossprod.HDF5Matrix, *85*, *124*

var, 125