Package: slam (via r-universe)

October 28, 2026

Version 0.1-56

Title Sparse Lightweight Arrays and Matrices

Description Data structures and algorithms for sparse arrays and matrices, based on index arrays and simple triplet representations, respectively.

Depends R (>= 3.4.0)

Imports stats

Enhances Matrix, SparseM, spam

License GPL-2

NeedsCompilation yes

Author Kurt Hornik [aut, cre] (ORCID: <https://orcid.org/0000-0003-4198-9911>), David Meyer [aut] (ORCID: <https://orcid.org/0000-0002-5196-3048>), Christian Buchta [aut]

Maintainer Kurt Hornik <Kurt.Hornik@R-project.org>

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

Date/Publication 2026-07-13 11:33:51 UTC

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

RemoteRef HEAD

RemoteSha e718b9bf09db98b0cdce93f4c513e730eaa95430

Contents

abind 2
apply 3
crossprod 4
foreign 5
norms 6
options 7
rollup 8
simple_sparse_array 10
simple_triplet_matrix 11
sums 13

Index **15**

`abind` *Combine Sparse Arrays*

Description

Combine a sequence of (sparse) arrays, matrices, or vectors into a single sparse array of the same or higher dimension.

Usage

```
abind_simple_sparse_array(..., MARGIN = 1L)
extend_simple_sparse_array(x, MARGIN = 0L)
```

Arguments

`...`	R objects of (or coercible to) class `simple_sparse_array`.
`MARGIN`	The dimension along which to bind the arrays.
`x`	An object of class `simple_sparse_array`.

Details

`abind_simple_sparse_array` automatically extends the dimensions of the elements of '`...`' before it combines them along the dimension specified in `MARGIN`. If a negative value is specified first all elements are extended left of the target dimension.

`extend_simple_sparse_array` inserts one (or more) one-level dimension(s) into `x` to the right of the position(s) specified in `MARGIN`, or to the left if specified in negative terms. Note that the target positions must all be in the range of the dimensions of `x` (see Examples).

Value

An object of class `simple_sparse_array` where the `dimnames` are taken from the elements of '`...`'.

Author(s)

Christian Buchta

See Also

`simple_sparse_array` for sparse arrays.

Examples

```
## automatic
abind_simple_sparse_array(1:3, array(4:6, c(1,3)))
abind_simple_sparse_array(1:3, array(4:6, c(3,1)), MARGIN = 2L)

## manual
abind_simple_sparse_array(1:3, 4:6)
abind_simple_sparse_array(1:3, 4:6, MARGIN = -2L)   ## by columns
abind_simple_sparse_array(1:3, 4:6, MARGIN = -1L)   ## by rows

##
a <- as.simple_sparse_array(1:3)
a
extend_simple_sparse_array(a, c( 0L, 1L))
extend_simple_sparse_array(a, c(-1L,-2L))   ## the same
extend_simple_sparse_array(a, c( 1L, 1L))
```

`apply` *Apply Functions Over Sparse Matrix Margins*

Description

Apply functions to (the cross-pairs of) the rows or columns of a sparse matrix.

Usage

```
rowapply_simple_triplet_matrix(x, FUN, ...)
colapply_simple_triplet_matrix(x, FUN, ...)

crossapply_simple_triplet_matrix(x, y = NULL, FUN, ...)
tcrossapply_simple_triplet_matrix(x, y = NULL, FUN, ...)
```

Arguments

x, y	a matrix in `simple_triplet_matrix`-form or, one of `x` and `y`, of class `matrix`.
FUN	the name of the function to be applied.
...	optional arguments to `FUN`.

Details

`colapply_simple_triplet_matrix` temporarily expands each column of `x` to dense `vector` representation and applies the function specified in `FUN`.

`crossapply_simple_triplet_matrix` temporarily expands each cross-pair of columns of `x` (and `y`) to dense `vector` representation and applies the function specified in `FUN`.

Note that if `y = NULL` then only the entries in the lower triangle and the diagonal are computed, assuming that `FUN` is symmetric.

Value

A `vector` (`matrix`) of length (dimensionality) of the margin(s) used. The type depends on the result of `FUN`.

Note that the result of `colapply_simple_triplet_matrix` is never simplified to `matrix`.

Author(s)

Christian Buchta

See Also

`apply` for dense-on-dense computations.

Examples

```
##
x <- matrix(c(1, 0, 0, 2, 1, 0), nrow = 3,
    dimnames = list(1:3, LETTERS[1:2]))
x
s <- as.simple_triplet_matrix(x)
colapply_simple_triplet_matrix(s, FUN = var)
##
simplify2array(colapply_simple_triplet_matrix(s, identity))
##
crossapply_simple_triplet_matrix(s, FUN = var)
```

`crossprod` *Matrix Crossproduct*

Description

Compute the matrix cross-product of a sparse and a dense or sparse matrix.

Usage

```
tcrossprod_simple_triplet_matrix(x, y = NULL)
##
crossprod_simple_triplet_matrix(x, y = NULL)
matprod_simple_triplet_matrix(x, y)
```

Arguments

`x, y`	a matrix in `simple_triplet_matrix`-form and/or a dense matrix, where one must be of the form indicated by the suffix.

Details

Function `tcrossprod_simple_triplet_matrix` implements fast computation of `x %*% t(x)` and `x %*% t(y)` (`tcrossprod`). The remaining functions are (optimized) wrappers.

Value

A double matrix, with appropriate `dimnames` taken from `x` and `y`.

Note

The computation is delegated to `tcrossprod` if `y` (or `x` if `y == NULL`) contains any of the special values `NA`, `NaN`, or `Inf`.

Author(s)

Christian Buchta

See Also

`crossprod` for dense-on-dense computations.

Examples

```
##
x <- matrix(c(1, 0, 0, 2, 1, 0), nrow = 3)
x
s <- as.simple_triplet_matrix(x)
tcrossprod_simple_triplet_matrix(s, x)
##
tcrossprod_simple_triplet_matrix(s)
##
tcrossprod_simple_triplet_matrix(s[1L, ], s[2:3, ])
```

---

foreign *Read and Write Sparse Matrix Format Files*

Description

Read and write CLUTO sparse matrix format files, or the CCS format variant employed by the MC toolkit.

Usage

```
read_stm_CLUTO(file)
write_stm_CLUTO(x, file)
read_stm_MC(file, scalingtype = NULL)
write_stm_MC(x, file)
```

Arguments

`file`	a character string with the name of the file to read or write.
`x`	a matrix object.
`scalingtype`	a character string specifying the type of scaling to be used, or `NULL` (default), in which case the scaling will be inferred from the names of the files with non-zero entries found (see **Details**).

Details

Documentation for CLUTO including its sparse matrix format used to be available from 'https://www-users.cse.umn.edu

`read_stm_CLUTO` reads CLUTO sparse matrices, returning a simple triplet matrix.

`write_stm_CLUTO` writes CLUTO sparse matrices. Argument `x` must be coercible to a simple triplet matrix via `as.simple_triplet_matrix`.

MC is a toolkit for creating vector models from text documents (see https://www.cs.utexas.edu/~dml/software/mc/). It employs a variant of Compressed Column Storage (CCS) sparse matrix format, writing data into several files with suitable names: e.g., a file with '`_dim`' appended to the base file name stores the matrix dimensions. The non-zero entries are stored in a file the name of which indicates the scaling type used: e.g., '`_tfx_nz`' indicates scaling by term frequency ('`t`'), inverse document frequency ('`f`') and no normalization ('`x`'). See '`README`' in the MC sources for more information.

`read_stm_MC` reads such sparse matrix information with argument `file` giving the path with the base file name, and returns a simple triplet matrix.

`write_stm_MC` writes matrices in MC CCS sparse matrix format. Argument `x` must be coercible to a simple triplet matrix via `as.simple_triplet_matrix`.

---

norms *Row and Column Norms*

Description

Compute row and column p-norms.

Usage

```
row_norms(x, p = 2)
col_norms(x, p = 2)
```

Arguments

`x`	a sparse `simple_triplet_matrix`, or a dense matrix.
`p`	a numeric at least one. Using `Inf` gives the maximum norm.

Value

A vector with the row or column p-norms for the given matrix.

Examples

```
x <- matrix(1 : 9, 3L)
## Row lengths:
row_norms(x)
## Column maxima:
col_norms(x, Inf)
```

`options` *Options for the 'slam' package*

Description

Function for getting and setting options for the **slam** package.

Usage

```
slam_options(option, value)
```

Arguments

`option`	character string indicating the option to get or set (see details). If missing, all options are returned as a list.
`value`	Value to be set. If omitted, the current value is returned.

Details

Currently, the following options are available:

`"max_dense"`: numeric specifying the maximum `length` of dense vectors (default: `2^24`).

See Also

`simple_sparse_array`

Examples

```
## save defaults
.slam_options <- slam_options()
.slam_options

slam_options("max_dense", 2^25)
slam_options("max_dense")

## reset
slam_options("max_dense", .slam_options$max_dense)
```

rollup *Rollup Sparse Arrays*

Description

Rollup (aggregate) sparse arrays along arbitrary dimensions.

Usage

```
rollup(x, MARGIN, INDEX, FUN, ...)

## S3 method for class 'simple_triplet_matrix'
rollup(x, MARGIN, INDEX = NULL, FUN = sum, ...,
       REDUCE = FALSE)
## S3 method for class 'simple_sparse_array'
rollup(x, MARGIN, INDEX = NULL, FUN = sum, ...,
       DROP = FALSE, EXPAND = c("none", "sparse", "dense", "all"),
       MODE = "double")
## S3 method for class 'matrix'
rollup(x, MARGIN, INDEX = NULL, FUN = sum, ...,
       DROP = FALSE, MODE = "double")
## S3 method for class 'array'
rollup(x, MARGIN, INDEX = NULL, FUN = sum, ...,
       DROP = FALSE, MODE = "double")
```

Arguments

x	a sparse (or dense) array, typically of numeric or logical values.
MARGIN	a vector giving the subscripts (names) of the dimensions to be rolled up.
INDEX	a corresponding (list of) factor (components) in the sense that as.factor defines the grouping(s). If NULL collapses the corresponding dimension(s) (default).
FUN	the name of the function to be applied.
REDUCE	option to remove zeros from the result.
DROP	option to delete the dimensions of the result which have only one level.
EXPAND	the cell expansion method to use (see Details).
MODE	the type to use for the values if the result is empty.
...	optional arguments to FUN.

Details

Provides aggregation of sparse and dense arrays, in particular fast summation over the rows or columns of sparse matrices in simple_triplet-form.

If (a component of) INDEX contains NA values the corresponding parts of x are omitted.

For simple_sparse_array the following cell expansion methods are provided:

`none`: The *non-zero* entries of a cell, if any, are supplied to `FUN` as a `vector`.

`sparse`: The number of *zero* entries of a cell is supplied in addition to above, as a second argument.

`dense`: Cells with *non-zero* entries are expanded to a dense `array` and supplied to `FUN`.

`all`: All cells are expanded to a dense `array` and supplied to `FUN`.

Note that the memory and time consumption increases with the level of expansion.

Note that the default method tries to coerce `x` to `array`.

Value

An object of the same class as `x` where for class `simple_triplet_matrix` the values are always of type `double` if `FUN = sum` (default).

The `dimnames` corresponding to `MARGIN` are based on (the components of) `INDEX`.

Note

Currently most of the code is written in R and, therefore, the memory and time it consumes is not optimal.

Author(s)

Christian Buchta

See Also

`simple_triplet_matrix` and `simple_sparse_array` for sparse arrays.

`apply` for dense arrays.

Examples

```
##
x <- matrix(c(1, 0, 0, 2, 1, NA), nrow = 2,
     dimnames = list(A = 1:2, B = 1:3))
x
apply(x, 1L, sum, na.rm = TRUE)
##
rollup(x, 2L, na.rm = TRUE)
##
rollup(x, 2L, c(1,2,1), na.rm = TRUE)
## omit
rollup(x, 2L, c(1,NA,1), na.rm = TRUE)
## expand
a <- as.simple_sparse_array(x)
a
r <- rollup(a, 1L, FUN = mean, na.rm = TRUE, EXPAND = "dense")
as.array(r)
##
r <- rollup(a, 1L, FUN = function(x, nz)
 length(x) / (length(x) + nz),
    EXPAND = "sparse"
```

```
)
as.array(r)
```

---

simple_sparse_array *Simple Sparse Arrays*

Description

Data structures and operators for sparse arrays based on a representation by index matrix and value vector.

Usage

```
simple_sparse_array(i, v, dim = NULL, dimnames = NULL)
simple_sparse_zero_array(dim, mode = "double")

as.simple_sparse_array(x)
is.simple_sparse_array(x)

simplify_simple_sparse_array(x, higher = TRUE)
reduce_simple_sparse_array(x, strict = FALSE, order = FALSE)
drop_simple_sparse_array(x)
```

Arguments

i	Integer matrix of array indices.
v	Vector of values.
dim	Integer vector specifying the size of the dimensions.
dimnames	either NULL or the names for the dimensions. This is a list with one component for each dimension, either NULL or a character vector of the length given by dim for that dimension. The list can be named, and the list names will be used as names for the dimensions. If the list is shorter than the number of dimensions, it is extended by NULL's to the length required.
mode	Character string specifying the mode of the values.
x	An R object; an object of class simple_sparse_array (see Note).
higher	Option to use the dimensions of the values (see Note).
strict	Option to treat violations of sparse representation as error (see Note).
order	Option to reorder elements (see Note).

Details

simple_sparse_array is a generator for a class of "lightweight" sparse arrays, represented by index matrices and value vectors. Currently, only methods for indexing and coercion are implemented.

simple_sparse_zero_array is a convenience function for the creation of empty arrays.

Note

The *zero* element is defined as `vector(typeof(v), 1L)`, for example, `FALSE` for `logical` values (see `vector`). Clearly, sparse arrays should not contain *zero* elements, however, for performance reasons the class generator does not remove them.

If `strict = FALSE` (default) `reduce_simple_sparse_array` tries to repair violations of sparse representation (*zero, multiple* elements), otherwise it stops. If `order = TRUE` the elements are further reordered (see `array`).

`simplify_simple_sparse_array` tries to reduce `v`. If `higher = TRUE` (default) augments `x` by the common dimensions of `v` (from the left), or the common length. Note that *scalar* elements are never extended and unused dimensions never dropped.

`drop_simple_sparse_array` drops unused dimensions.

If `prod(dim(x)) > slam_options("max_dense")` empty and negative indexing are disabled for `[` and `[<-`. Further, non-negative single (vector) indexing is limited to 52 bits of representation.

See Also

`simple_triplet_matrix` for sparse matrices.

`slam_options` for options.

Examples

```
x <- array(c(1, 0, 0, 2, 0, 0, 0, 3), dim = c(2, 2, 2))
s <- as.simple_sparse_array(x)
identical(x, as.array(s))

simple_sparse_array(matrix(c(1, 3, 1, 3, 1, 3), nrow = 2), c(1, 2))
```

---

simple_triplet_matrix *Simple Triplet Matrix*

Description

Data structures and operators for sparse matrices based on simple triplet representation.

Usage

```
simple_triplet_matrix(i, j, v, nrow = max(i), ncol = max(j),
                      dimnames = NULL)
simple_triplet_zero_matrix(nrow, ncol = nrow, mode = "double")
simple_triplet_diag_matrix(v, nrow = length(v))

as.simple_triplet_matrix(x)
is.simple_triplet_matrix(x)
```

Arguments

i, j	Integer vectors of row and column indices, respectively.
v	Vector of values.
nrow, ncol	Integer values specifying the number of rows and columns, respectively. Defaults are the maximum row and column indices, respectively.
dimnames	A dimnames attribute for the matrix: NULL or a list of length 2 giving the row and column names respectively. An empty list is treated as NULL, and a list of length one as row names. The list can be named, and the list names will be used as names for the dimensions.
mode	Character string specifying the mode of the values.
x	An R object.

Details

simple_triplet_matrix is a generator for a class of "lightweight" sparse matrices, "simply" represented by triplets (i, j, v) of row indices i, column indices j, and values v, respectively. simple_triplet_zero_matrix and simple_triplet_diag_matrix are convenience functions for the creation of empty and diagonal matrices.

Currently implemented operations include the addition, subtraction, multiplication and division of compatible simple triplet matrices, as well as the multiplication and division of a simple triplet matrix and a vector. Comparisons of the elements of a simple triplet matrices with a number are also provided. In addition, methods for indexing, combining by rows (rbind) and columns (cbind), transposing (t), concatenating (c), and detecting/extracting duplicated and unique rows are implemented.

See Also

simple_sparse_array for sparse arrays.

Examples

```
x <- matrix(c(1, 0, 0, 2), nrow = 2)
s <- as.simple_triplet_matrix(x)
identical(x, as.matrix(s))

simple_triplet_matrix(c(1, 4), c(1, 2), c(1, 2))
simple_triplet_zero_matrix(3)
simple_triplet_diag_matrix(1:3)

cbind(rbind(s, t(s)), rbind(s, s))
## Not run:
## map to default Matrix class
stopifnot(require("Matrix"))
sparseMatrix(i = s$i, j = s$j, x = s$v, dims = dim(s),
      dimnames = dimnames(s))

## End(Not run)
```

sums *Form Row and Column Sums and Means*

Description

Form row and column sums and means for sparse arrays (currently `simple_triplet_matrix` only).

Usage

```
row_sums(x, na.rm = FALSE, dims = 1, ...)
col_sums(x, na.rm = FALSE, dims = 1, ...)
row_means(x, na.rm = FALSE, dims = 1, ...)
col_means(x, na.rm = FALSE, dims = 1, ...)

## S3 method for class 'simple_triplet_matrix'
row_sums(x, na.rm = FALSE, dims = 1, ...)
## S3 method for class 'simple_triplet_matrix'
col_sums(x, na.rm = FALSE, dims = 1, ...)
## S3 method for class 'simple_triplet_matrix'
row_means(x, na.rm = FALSE, dims = 1, ...)
## S3 method for class 'simple_triplet_matrix'
col_means(x, na.rm = FALSE, dims = 1, ...)
```

Arguments

`x`	a sparse array containing numeric, integer, or logical values.
`na.rm`	logical. Should missing values (including `NaN`) be omitted from the calculations?
`dims`	currently not used for sparse arrays.
`...`	currently not used for sparse arrays.

Details

Provides fast summation over the rows or columns of sparse matrices in `simple_triplet`-form.

Value

A numeric (double) array of suitable size, or a vector if the result is one-dimensional. The `dimnames` (or `names` for a vector result) are taken from the original array.

Note

Results are always of storage type `double` to avoid (integer) overflows.

Author(s)

Christian Buchta

See Also

simple_triplet_matrix, colSums for dense numeric arrays.

Examples

```
##
x <- matrix(c(1, 0, 0, 2, 1, NA), nrow = 3)
x
s <- as.simple_triplet_matrix(x)
row_sums(s)
row_sums(s, na.rm = TRUE)
col_sums(s)
col_sums(s, na.rm = TRUE)
```

Index

∗ **IO**
foreign, 5
∗ **algebra**
apply, 3
crossprod, 4
rollup, 8
sums, 13
∗ **arith**
rollup, 8
sums, 13
∗ **array**
abind, 2
apply, 3
crossprod, 4
rollup, 8
sums, 13
∗ **math**
options, 7
simple_sparse_array, 10
simple_triplet_matrix, 11

abind, 2
abind_simple_sparse_array *(*abind*)*, 2
apply, 3, *4*, *9*
array, *11*
as.simple_sparse_array
*(*simple_sparse_array*)*, 10
as.simple_triplet_matrix, *6*
as.simple_triplet_matrix
*(*simple_triplet_matrix*)*, 11

col_means *(*sums*)*, 13
col_norms *(*norms*)*, 6
col_sums *(*sums*)*, 13
colapply_simple_triplet_matrix *(*apply*)*,
3
colSums, *14*
crossapply_simple_triplet_matrix
*(*apply*)*, 3
crossprod, 4, *5*
crossprod_simple_triplet_matrix
*(*crossprod*)*, 4

drop_simple_sparse_array
*(*simple_sparse_array*)*, 10

extend_simple_sparse_array *(*abind*)*, 2

foreign, 5

is.simple_sparse_array
*(*simple_sparse_array*)*, 10
is.simple_triplet_matrix
*(*simple_triplet_matrix*)*, 11

matprod_simple_triplet_matrix
*(*crossprod*)*, 4

norms, 6

options, 7

read_stm_CLUTO *(*foreign*)*, 5
read_stm_MC *(*foreign*)*, 5
reduce_simple_sparse_array
*(*simple_sparse_array*)*, 10
rollup, 8
row_means *(*sums*)*, 13
row_norms *(*norms*)*, 6
row_sums *(*sums*)*, 13
rowapply_simple_triplet_matrix *(*apply*)*,
3

simple sparse array
*(*simple_sparse_array*)*, 10
simple triplet matrix, *6*
simple triplet matrix
*(*simple_triplet_matrix*)*, 11
simple_sparse_array, *2*, *7*, *9*, 10, *12*
simple_sparse_zero_array
*(*simple_sparse_array*)*, 10

simple_triplet_diag_matrix
(simple_triplet_matrix), 11
simple_triplet_matrix, 6, 9, 11, 11
simple_triplet_zero_matrix
(simple_triplet_matrix), 11
simplify_simple_sparse_array
(simple_sparse_array), 10
slam_options, 11
slam_options (options), 7
sums, 13

tcrossapply_simple_triplet_matrix
(apply), 3
tcrossprod_simple_triplet_matrix
(crossprod), 4

vector, 11

write_stm_CLUTO (foreign), 5
write_stm_MC (foreign), 5