Package 'MatrixExtra'

May 15, 2022

Type Package

Title Extra Methods for Sparse Matrices

Version 0.1.10

Maintainer David Cortes <david.cortes.rivera@gmail.com>

Description Extends sparse matrix and vector classes from the 'Matrix' package by providing:
(a) Methods and operators that work natively on CSR formats (compressed sparse row, a.k.a. 'RsparseMatrix') such as slicing/sub-setting, assignment, rbind(), mathematical operators for CSR and COO such as addition (``+") or sqrt(), and methods such as diag();
(b) Multi-threaded matrix multiplication and cross-product for many <sparse, dense> types, including the 'float32' type from 'float';
(c) Coercion methods between pairs of classes which are not present in 'Matrix', such as 'dgCMatrix' -> 'ngRMatrix', as well as convenience conversion functions;
(d) Utility functions for sparse matrices such as sorting the indices or removing zero-valued entries;
(e) Fast transposes that work by outputting in the opposite storage format;
(f) Faster replacements for many 'Matrix' methods for all sparse types, such as slicing and elementwise multiplication.
(g) Convenience functions for sparse objects, such as 'mapSparse' or a shorter 'show' method.

License GPL (>= 2)

Depends Matrix (>= 1.3), methods

Imports Rcpp, RhpcBLASctl, float

LinkingTo Rcpp, float

Suggests testthat, data.table, knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/david-cortes/MatrixExtra

BugReports https://github.com/david-cortes/MatrixExtra/issues

RoxygenNote 7.1.2

NeedsCompilation yes

Biarch TRUE

StagedInstall TRUE

Author David Cortes [aut, cre, cph],
Dmitry Selivanov [cph],
Thibaut Goetghebuer-Planchon [cph] (Copyright holder of included robinmap library),
Martin Maechler [cph] (Copyright holder of 'Matrix' package from which some code was taken),
Robert Gentleman [cph] (Copyright holder of mathematical functions used by base R which were copied),
Ross Ihaka [cph] (Copyright holder of mathematical functions used by base R which were copied)

Repository CRAN

Date/Publication 2022-05-15 18:30:02 UTC

R topics documented:

assignment *Assignment operator for CSR matrices*

Description

Assign values to a CSR matrix. Note: this will only be a relatively fast operation when assigning contiguous row sequences. Only some of the potential assignment cases to a CSR matrix are replaced here - for example, cases that involve uneven recycling of vectors will be left to the 'Matrix' package.

Usage

```
## S4 replacement method for signature 'dgRMatrix,index,index,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'dgRMatrix,missing,index,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'dgRMatrix,index,missing,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'dgRMatrix,missing,missing,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'dgRMatrix,index,index,sparseVector'
x[i, j, ...] <- value

## S4 replacement method for signature 'dgRMatrix,missing,index,sparseVector'
x[i, j, ...] <- value

## S4 replacement method for signature 'dgRMatrix,index,missing,sparseVector'
x[i, j, ...] <- value

## S4 replacement method for signature 'dgRMatrix,missing,missing,sparseVector'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,nsparseVector,nsparseVector,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,missing,nsparseVector,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,nsparseVector,missing,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,lsparseVector,lsparseVector,replValue'
x[i, j, ...] <- value
```

```
## S4 replacement method for signature 'ANY,missing,lsparseVector,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,lsparseVector,missing,replValue'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,nsparseVector,nsparseVector,ANY'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,missing,nsparseVector,ANY'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,nsparseVector,missing,ANY'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,lsparseVector,lsparseVector,ANY'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,missing,lsparseVector,ANY'
x[i, j, ...] <- value

## S4 replacement method for signature 'ANY,lsparseVector,missing,ANY'
x[i, j, ...] <- value
```

Arguments

x	A CSR matrix whose values are to be replaced.
i	The indices of the rows to replace.
j	The indices of the columns to replace.
...	Not used
value	The values to replace with.

Value

The same ‘x‘ input with the values ‘[i,j]‘ set to ‘value‘. If the result is a full matrix (e.g. ‘x[,] <- 1‘), the object will be a dense matrix from base R.

Examples

```
library(Matrix)
library(MatrixExtra)
set.seed(1)
X <- rsparsematrix(5, 3, .5, repr="R")
X[1:3] <- 0
print(X)
```

cbind2-method *Concatenate sparse matrices by columns*

Description

'cbind2' method for the CSR and COO sparse matrix and sparse vector classes from 'Matrix', taking the most efficient route for the concatenation according to the input types.

Usage

```
## S4 method for signature 'TsparseMatrix,TsparseMatrix'
cbind2(x, y)

## S4 method for signature 'TsparseMatrix,sparseVector'
cbind2(x, y)

## S4 method for signature 'sparseVector,TsparseMatrix'
cbind2(x, y)

## S4 method for signature 'CsparseMatrix,sparseVector'
cbind2(x, y)

## S4 method for signature 'sparseVector,CsparseMatrix'
cbind2(x, y)

## S4 method for signature 'sparseVector,sparseVector'
cbind2(x, y)

## S4 method for signature 'RsparseMatrix,RsparseMatrix'
cbind2(x, y)

## S4 method for signature 'TsparseMatrix,RsparseMatrix'
cbind2(x, y)

## S4 method for signature 'RsparseMatrix,TsparseMatrix'
cbind2(x, y)

## S4 method for signature 'RsparseMatrix,numeric'
cbind2(x, y)

## S4 method for signature 'RsparseMatrix,integer'
cbind2(x, y)

## S4 method for signature 'RsparseMatrix,logical'
cbind2(x, y)

## S4 method for signature 'RsparseMatrix,sparseVector'
```

```
cbind2(x, y)

## S4 method for signature 'numeric,RsparseMatrix'
cbind2(x, y)

## S4 method for signature 'integer,RsparseMatrix'
cbind2(x, y)

## S4 method for signature 'logical,RsparseMatrix'
cbind2(x, y)

## S4 method for signature 'sparseVector,RsparseMatrix'
cbind2(x, y)
```

Arguments

x	First matrix to concatenate.
y	Second matrix to concatenate.

Value

A sparse matrix (storage order varying depending on the input types).

Examples

```
library(Matrix)
library(MatrixExtra)
set.seed(1)
X <- rsparsematrix(3, 4, .3)
X <- as(X, "TsparseMatrix")
inherits(cbind2(X, X), "TsparseMatrix")
```

---

check_sparse_matrix *Check if the underlying data behind a sparse matrix constitutes a valid object*

Description

Check if the underlying data behind a sparse matrix constitutes a valid object

Usage

```
check_sparse_matrix(X, sort = TRUE, remove_zeros = TRUE)
```

Arguments

X	A sparse matrix or sparse vector whose underlying arrays are to be checked.
sort	Whether to sort the indices of 'X' along the way. For this, will make deep copies of the indices and values so that there are no issues with external references being updated along the way.
remove_zeros	Whether to remove entries in 'X' which have a value of zero but are nevertheless still present in the sparse representation.

Details

Makes checks on the data contained in the sparse matrix or sparse vector object for whether the data constitutes a valid matrix - e.g. indices must not be negative or larger than the dimensions, index pointer must match with the indices, etc.

As a short-hand, can also sort the matrix and remove zeros by calling the respective functions sort_sparse_indices and remove_sparse_zeros.

A sparse matrix or sparse vector should never come out with invalid data from functions from 'Matrix' and 'MatrixExtra', with one exception: some 'MatrixExtra' functions might modify indices in-place, which can cause problems if the same array of indices / values is also used by another matrix or R object elsewhere.

Otherwise, this function is aimed at making checks on matrices that are manually constructed.

Value

The same matrix or vector 'X' (perhaps sorted or with zeros removed depending on the parameters). If 'X' contains data that doesn't make for a valid sparse matrix (e.g. different number of values and indices), it will throw an error.

---

conversions *Conversions between matrix types*

Description

Convenience functions for converting to different sparse matrix formats, between pairs of classes which might not be supported in the 'Matrix' package.

These come in the form of explicit functions 'as.<type>.matrix' (see below), as well as registered conversion methods to use along with 'as(object, type)', adding extra conversion routes which are missing in the 'Matrix' package for output types 'dgRMatrix', 'lgRMatrix', and 'ngRMatrix'.

Usage

```
as.csr.matrix(x, binary = FALSE, logical = FALSE, sort = FALSE)

as.csc.matrix(x, binary = FALSE, logical = FALSE, sort = FALSE)

as.coo.matrix(x, binary = FALSE, logical = FALSE, sort = FALSE)

as.sparse.vector(x, binary = FALSE, logical = FALSE, integer = FALSE)
```

Arguments

x	A matrix which is to be converted to a different format. Supported input types are: • Sparse matrices from ‘Matrix‘ package, in any format. • Sparse vectors from ‘Matrix‘ in any format. • Dense matrices from base R (class ‘matrix‘). • Dense vectors from base R (classes ‘numeric‘, ‘integer‘, ‘logical‘). • Dense matrix or vector from package ‘float‘ (class ‘float32‘). • ‘data.frame‘, ‘data.table‘, and ‘tibble‘.
binary	Whether the result should be a binary-only matrix/vector (inheriting from class ‘nsparseMatrix‘/‘nsparseVector‘ - these don’t have slot ‘x‘). Can only pass one of ‘binary‘ or ‘logical‘.
logical	Whether the result should be a matrix/vector with logical (boolean) type (inheriting from ‘lsparseMatrix‘/‘lsparseVector‘). Can only pass one of ‘binary‘ or ‘logical‘.
sort	Whether to sort the indices in case they are not sorted. Note that it will perform deep copies of the indices and values along the way.
integer	Whether the result should be a vector with integer type (’isparseVector’).

Details

The functions internally might use some routes of ‘as(x, "?sparseMatrix")‘, so they might work with other object classes if they register a conversion method for ‘Matrix‘ base types.

When passed a vector, the functions ‘as.csr.matrix‘ and ‘as.coo.matrix‘ will assume that it is a row vector, while ‘as.csc.matrix‘ will assume it’s a column vector.

Value

A sparse matrix/vector, with format:

- CSR (a.k.a. ‘RsparseMatrix‘) when calling ‘as.csr.matrix‘ (class ‘dgRMatrix‘, ‘ngRMatrix‘, or ‘lgRMatrix‘, depending on parameters ‘binary‘ and ‘logical‘).
- CSC (a.k.a. ‘CsparseMatrix‘) when calling ‘as.csc.matrix‘ (class ‘dgCMatrix‘, ‘ngCMatrix‘, or ‘lgCMatrix‘, depending on parameters ‘binary‘ and ‘logical‘).
- COO (a.k.a. ‘TsparseMatrix‘) when calling ‘as.coo.matrix‘ (class ‘dgTMatrix‘, ‘ngTMatrix‘, or ‘lgTMatrix‘, depending on parameters ‘binary‘ and ‘logical‘).
- sparse vector (class dependant on input) when calling ‘as.sparse.vector‘.

Examples

```
library(Matrix)
library(MatrixExtra)

m.coo <- as(matrix(1:3), "TsparseMatrix")
as.csr.matrix(m.coo)
as.csr.matrix(1:3) # <- assumes it's a row vector
```

```
as.csc.matrix(1:3) # <- assumes it's a column vector

### Using the new conversion methods
### (these would fail if 'MatrixExtra' is not loaded)
as(matrix(1:3), "ngRMatrix")
as(as.csc.matrix(m.coo), "dgRMatrix")
```

---

csr-linalg *Linear Algebra functions for CSR matrices*

Description

Short wrappers around some linear algebra operators from ‘Matrix‘ that take CSC matrices, adapted to work for CSR matrices without involving any data duplication or deep format conversion, thus saving time and memory.

Usage

```
## S4 method for signature 'RsparseMatrix,character'
norm(x, type = "O", ...)

## S4 method for signature 'RsparseMatrix,missing'
norm(x, type = "O", ...)

## S4 method for signature 'RsparseMatrix'
diag(x)

## S4 replacement method for signature 'RsparseMatrix'
diag(x) <- value
```

Arguments

x	A sparse matrix in CSR format.
type	Type of the norm to calculate (see norm).
...	Extra arguments to pass to ‘norm‘
value	Replacement value for the matrix diagonal.

Value

The same value that ‘Matrix‘ would return for CSC matrices.

deepcopy_sparse_object *Deep copy sparse matrices and vectors*

Description

Generates a deep copy of a sparse matrix or sparse vector object, which can come useful when the matrix is to later be passed to functions that will potentially modify it in-place, such as sort_sparse_indices.

Usage

```
deepcopy_sparse_object(X)
```

Arguments

X	A sparse matrix or sparse vector from the ‘Matrix‘ package.

Value

The same input ‘X‘ with the fields replaced with deep copies.

emptySparse *Create Empty Sparse Matrix*

Description

Creates an empty sparse matrix (all values being zeros) with the requested format and dimensions. This is a faster alternative to calling ‘Matrix::Matrix(0, ...)‘.

Usage

```
emptySparse(nrow = 0L, ncol = 0L, format = "R", dtype = "d")
```

Arguments

nrow	Desired number of rows for the matrix.
ncol	Desired number of columns for the matrix.
format	Storage format for the matrix. Options are: • "R", which will output a CSR Matrix ("RsparseMatrix"). • "C", which will output a CSC Matrix ("CsparseMatrix"). • "T", which will output a COO/triplets Matrix ("TsparseMatrix").
dtype	Data type for the matrix. Options are: • "d", which will output a numeric/double type (e.g. "dgRMatrix"). • "l", which will output a logical/boolean type. • "n", which will output a binary type.

Value

A sparse matrix of general type, with the specific class determined by ‘format‘ and ‘dtype‘.

Examples

```
### This is very fast despite the large dimensions,
### as no data is held in the resulting object
library(MatrixExtra)
X <- emptySparse(nrow=2^20, ncol=2^25, format="T")
```

`filterSparse` *Filter values of a sparse matrix or vector*

Description

Filters the non-zero values of a sparse matrix or sparse vector object according to a user-provided function (e.g. to take only values above a certain threshold, or only greater than the mean), returning a sparse object with the same dimension as the input, but having only the non-zero values that meet the desired criteria.

Usage

```
filterSparse(X, fn, ...)
```

Arguments

`X`	A sparse matrix or sparse vector.
`fn`	A function taking as first argument a vector of non-zero values (which will be extracted from ‘X‘) and returning a logical/boolean vector of the same length as the first argument, returning ‘TRUE‘ for values that are to be kept and ‘FALSE‘ for values that are to be discarded. Alternatively, can pass a logical/boolean vector of the same length as ‘X@x‘. If any of the returned values is ‘NA‘, will put a ‘NA‘ value at that position.
`...`	Extra arguments to pass to ‘fn‘.

Value

A sparse matrix or sparse vector of the same class as ‘X‘ and with the same dimensions, but having only the non-zero values that meet the condition specificed by ‘fn‘.

Examples

```
library(Matrix)
library(MatrixExtra)

### Random sparse matrix
set.seed(1)
X <- rsparsematrix(nrow=20, ncol=10, density=0.3)

### Take only values above 0.5
X_filtered <- filterSparse(X, function(x) x >= 0.5)

### Only elements with absolute values less than 0.3
X_filtered <- filterSparse(X, function(x) abs(x) <= 0.3)

### Only values above the mean (among non-zeros)
X_filtered <- filterSparse(X, function(x) x > mean(x))
```

---

mapSparse *Map values of a sparse matrix/vector*

Description

Applies a function to the non-zero values of a sparse object, returning the transformed input with the new non-zero values.

Usage

```
mapSparse(X, fn, ...)
```

Arguments

X	A sparse matrix or sparse vector, whose non-zero values will be transformed/mapped according to ‘fn‘.
fn	A function taking as first argument a vector of non-zero values (which will be extracted from ‘X‘) and returning another vector of the same length as the first argument, which will become the non-zero values of the output. Alternatively, can pass a vector of the same length as ‘X@x‘. If the results are of a different type than the input (e.g. input is ‘lsparseMatrix‘, but ‘fn‘ returns a numeric vector), the type will be automatically converted to match the type returned by ‘fn‘.
...	Extra arguments to pass to ‘fn‘.

Value

A sparse object with the same storage order (T/C/R), dimension, and number of non-zero entries as ‘X‘, but with its non-zero values substituted by the output from ‘fn‘, and the exact class determined by the type returned by ‘fn‘.

Examples

```
library(Matrix)
library(MatrixExtra)

set.seed(1)
X <- rsparsematrix(10, 5, .5)
print(mapSparse(X, function(x) abs(x)+1))
```

`mathematical-functions`

Mathematical functions for CSR and COO matrices

Description

Implements some mathematical functions for CSR (a.k.a. "RsparseMatrix") and COO (a.k.a. "TsparseMatrix") matrix types without converting them to CSC matrices in the process.

The functions here reduce to applying the same operation over the non-zero elements only, and as such do not benefit from any storage format conversion as done implicitly in the ‘Matrix‘ package.

Usage

```
## S4 method for signature 'RsparseMatrix'
sqrt(x)

## S4 method for signature 'TsparseMatrix'
sqrt(x)

## S4 method for signature 'RsparseMatrix'
abs(x)

## S4 method for signature 'TsparseMatrix'
abs(x)

## S4 method for signature 'RsparseMatrix'
log1p(x)

## S4 method for signature 'TsparseMatrix'
log1p(x)

## S4 method for signature 'RsparseMatrix'
sin(x)

## S4 method for signature 'TsparseMatrix'
sin(x)

## S4 method for signature 'RsparseMatrix'
tan(x)
```

```
## S4 method for signature 'TsparseMatrix'
tan(x)

## S4 method for signature 'RsparseMatrix'
tanh(x)

## S4 method for signature 'TsparseMatrix'
tanh(x)

## S4 method for signature 'RsparseMatrix'
tanpi(x)

## S4 method for signature 'TsparseMatrix'
tanpi(x)

## S4 method for signature 'RsparseMatrix'
sinh(x)

## S4 method for signature 'TsparseMatrix'
sinh(x)

## S4 method for signature 'RsparseMatrix'
atanh(x)

## S4 method for signature 'TsparseMatrix'
atanh(x)

## S4 method for signature 'RsparseMatrix'
expm1(x)

## S4 method for signature 'TsparseMatrix'
expm1(x)

## S4 method for signature 'RsparseMatrix'
sign(x)

## S4 method for signature 'TsparseMatrix'
sign(x)

## S4 method for signature 'RsparseMatrix'
ceiling(x)

## S4 method for signature 'TsparseMatrix'
ceiling(x)

## S4 method for signature 'RsparseMatrix'
floor(x)
```

```
## S4 method for signature 'TsparseMatrix'
floor(x)

## S4 method for signature 'RsparseMatrix'
trunc(x)

## S4 method for signature 'TsparseMatrix'
trunc(x)

## S4 method for signature 'RsparseMatrix'
round(x, digits = 0)

## S4 method for signature 'TsparseMatrix'
round(x, digits = 0)

## S4 method for signature 'RsparseMatrix'
signif(x, digits = 6)

## S4 method for signature 'TsparseMatrix'
signif(x, digits = 6)
```

Arguments

x	A CSR or COO matrix.
digits	See round and signif. If passing more than one value, will call the corresponding function from the ‘Matrix‘ package, which implies first converting ‘x‘ to CSC format.

Value

A CSR or COO matrix depending on the input. They will be of the ‘dg‘ type (‘dgRMatrix‘ or ‘dgTMatrix‘).

Examples

```
library(Matrix)
library(MatrixExtra)
options("MatrixExtra.quick_show" = FALSE)
set.seed(1)
X <- as.csr.matrix(rsparsematrix(4, 3, .4))
abs(X)
sqrt(X^2)
### This will output CSC
round(X, 1:2)
```

matmult *Multithreaded Sparse-Dense Matrix and Vector Multiplications*

Description

Multithreaded <matrix, matrix> multiplications ('%*%', 'crossprod', and 'tcrossprod') and <matrix, vector> multiplications ('%*%'), for <sparse, dense> matrix combinations and <sparse, vector> combinations (See signatures for supported combinations).

Objects from the 'float' package are also supported for some combinations.

Usage

```
## S4 method for signature 'matrix,CsparseMatrix'
x %*% y

## S4 method for signature 'float32,CsparseMatrix'
x %*% y

## S4 method for signature 'matrix,RsparseMatrix'
tcrossprod(x, y)

## S4 method for signature 'float32,RsparseMatrix'
tcrossprod(x, y)

## S4 method for signature 'matrix,CsparseMatrix'
crossprod(x, y)

## S4 method for signature 'float32,CsparseMatrix'
crossprod(x, y)

## S4 method for signature 'RsparseMatrix,matrix'
tcrossprod(x, y)

## S4 method for signature 'RsparseMatrix,matrix'
x %*% y

## S4 method for signature 'RsparseMatrix,float32'
x %*% y

## S4 method for signature 'RsparseMatrix,float32'
tcrossprod(x, y)

## S4 method for signature 'RsparseMatrix,numeric'
x %*% y

## S4 method for signature 'RsparseMatrix,logical'
```

```
x %*% y

## S4 method for signature 'RsparseMatrix,integer'
x %*% y

## S4 method for signature 'RsparseMatrix,sparseVector'
x %*% y
```

Arguments

x, y	dense (matrix / float32) and sparse (RsparseMatrix / CsparseMatrix) matrices or vectors (sparseVector, numeric, integer, logical).

Details

Will try to use the maximum available number of threads for the computations when appropriate. The number of threads can be controlled through the package options (e.g. ‘options("MatrixExtra.nthreads" = 1)‘ - see MatrixExtra-options) and will be set to 1 after running restore_old_matrix_behavior.

Be aware that sparse-dense matrix multiplications might suffer from reduced numerical precision, especially when using objects of type ‘float32‘ (from the ‘float‘ package).

Internally, these functions use BLAS level-1 routines, so their speed might depend on the BLAS backend being used (e.g. MKL, OpenBLAS) - that means: they might be quite slow on a default install of R for Windows (see this link for a tutorial about getting OpenBLAS in R for Windows).

Doing computations in float32 precision depends on the package float, and as such comes with some caveats:

- On Windows, if installing ‘float‘ from CRAN, it will use very unoptimized routines which will likely result in a slowdown compared to using regular double (numeric) type. Getting it to use an optimized BLAS library is not as simple as substituting the Rblas DLL - see the package’s README for details.
- On macOS, it will use static linking for ‘float‘, thus if changing the BLAS library used by R, it will not change the float32 functions, and getting good performance out of it might require compiling it from source with ‘-march=native‘ flag.

When multiplying a sparse matrix by a sparse vector, their indices will be sorted in-place (see sort_sparse_indices).

In order to match exactly with base R’s behaviors, when passing vectors to these operators, will assume their shape as follows:

- MatMult(Matrix, vector): column vector if the matrix has more than one column or is empty, row vector if the matrix has only one column.
- MatMult(vector, Matrix): row vector if the matrix has more than one row, column vector if the matrix has only one row.
- MatMul(vector, vector): LHS is a row vector, RHS is a column vector.
- crossprod(Matrix, vector): column vector if the matrix has more than one row, row vector if the matrix has only one row.
- crossprod(vector, Matrix): column vector.

- crossprod(vector, vector): column vector.
- tcrossprod(Matrix, vector): row vector if the matrix has only one row, column vector if the matrix has only one column, and will throw an error otherwise.
- tcrossprod(vector, Matrix): row vector if the matrix has more than one column, column vector if the matrix has only one column.
- tcrossprod(vector, vector): column vector.

In general, the output returned by these functions will be a dense matrix from base R, or a dense matrix from 'float' when one of the inputs is also from the 'float' package, with the following exceptions:

- MatMult(RsparseMatrix[n,1], vector) -> 'dgRMatrix'.
- MatMult(RsparseMatrix[n,1], sparseVector) -> 'dgCMatrix'.
- MatMult(float32[n], CsparseMatrix[1,m]) -> 'dgCMatrix'.
- tcrossprod(float32[n], RsparseMatrix[m,1]) -> 'dgCMatrix'.

Value

A dense `matrix` object in most cases, with some exceptions which might come in sparse format (see the 'Details' section).

Examples

```
library(Matrix)
library(MatrixExtra)
### To use all available threads (default)
options("MatrixExtra.nthreads" = parallel::detectCores())
### Example will run with only 1 thread (CRAN policy)
options("MatrixExtra.nthreads" = 1)

## Generate random matrices
set.seed(1)
A <- rsparsematrix(5,4,.5)
B <- rsparsematrix(4,3,.5)

## Now multiply in some supported combinations
as.matrix(A) %*% as.csc.matrix(B)
as.csr.matrix(A) %*% as.matrix(B)
crossprod(as.matrix(B), as.csc.matrix(B))
tcrossprod(as.csr.matrix(A), as.matrix(A))

### Restore the number of threads
options("MatrixExtra.nthreads" = parallel::detectCores())
```

`MatrixExtra` *MatrixExtra package*

Description

Additional methods for the sparse matrices and sparse vector classes from the 'Matrix' package, with an emphasis on the CSR (compressed sparse row) format (a.k.a. "RsparseMatrix" in 'Matrix' parlance).

This package provides, among others:

- Fast and multi-threaded matrix multiplications for various combinations of '<sparse,dense>' and '<dense,sparse>' types (see matmult), including the dense 'float32' types from the 'float' package (see float-package).
- Operations that work efficiently in the CSR format, such as concatenating by rows (see rbind2-method) or selecting rows (see slice).
- Convenience conversion functions for the different sparse matrix types (see conversions).
- Overloaded operators such as "+" or "*" for many combinations of '<RsparseMatrix,RsparseMatrix>', '<RsparseMatrix,TsparseMatrix>', '<RsparseMatrix,matrix>', and '<RsparseMatrix,scalar>' (see operators).
- Overloaded mathematical functions that work only on the non-zero entries of matrices (see mathematical-functions).
- Fast transposes which change the storage format (see t_shallow).
- Utility functions for sparse matrices (see e.g. sort_sparse_indices and remove_sparse_zeros).
- Faster replacements for many methods from 'Matrix' for all the sparse formats (COO, CSR, CSC), such as '[' (slice), addition ('+'), elementwise multiplication ('*'), among others (see operators).
- Convenience functions for sparse objects, such as mapSparse, emptySparse, and a shorter show method for sparse objects.

Important: 'MatrixExtra' modifies some important behaviors from the 'Matrix' library, which might cause some functions from 'Matrix' or other packages to stop working under the default options of 'MatrixExtra'. See MatrixExtra-options for details.

`MatrixExtra-options` *MatrixExtra internal options*

Description

Controls some of the behaviors when calling methods from ‘MatrixExtra‘ which differ very significantly from the same methods in ‘Matrix‘ and which have the potential for breaking existing code in other packages.

For example, the function ‘Matrix::sparse.model.matrix‘ might throw errors if it is called after loading ‘library(MatrixExtra)‘ with the default options, and the default transpose behavior needs to be modified from ‘MatrixExtra‘ to make it work again.

The ‘Matrix‘ package has some particular choice of behaviors which can inadvertently make operations very inefficient. Particularly:

- When transposing a sparse matrix in CSR or CSC formats, ‘Matrix‘ will output a transpose in the same storage order. This is a slow operation, as it requires duplicating the data, creating a new index, and sorting it. In general, one might not care about the storage order of a sparse matrix until it comes the time of performing an operation which is more efficient in one format, or one might want to pass a sparse matrix object to a function from another package which is not under one’s control, for which a slow ‘t(.)‘ is undesirable.

  ‘MatrixExtra‘ will instead make the default for ‘t(CSR)‘ and ‘t(CSC)‘ to output in the opposite format (‘t(CSR)‘ -> ‘CSC‘; ‘t(CSC‘ -> ‘CSR‘), which does not involve any changes or duplication in the data, and it’s thus much faster. This in particular could break code in existing packages, particularly in **the ‘Matrix‘ package itself**. The old ‘Matrix‘ behavior can be brought back through ‘options("MatrixExtra.fast_transpose" = FALSE)‘.

  For example, the function ‘Matrix::sparse.model.matrix‘ is unlikely to work under the default fast transpose option, and getting it to work again will require setting ‘options("MatrixExtra.fast_transpose" = FALSE)‘ or calling ‘MatrixExtra::restore_old_matrix_behavior()‘.

- When selecting slices of a sparse matrix with only one row or only one column, ‘Matrix‘ will by default simplify them to a **dense** vector, which goes against the purpose of using sparse structures. With this being the default, it’s possible to inadvertently do this and end up losing all the benefits of a sparse data structure.

  ‘MatrixExtra‘ instead will make the default to simplify them to a **sparse** vector (see slice). It can be changed back to the ‘Matrix‘ behavior through ‘options("MatrixExtra.drop_sparse" = FALSE)‘.

- When doing some operations which require the indices of sparse matrices to be sorted, such as elementwise addition and multiplication, ‘Matrix‘ will create deep copies of the indices and values, which takes extra time but ensures that no external references to the same objects are impacted, thus playing well with R’s pass-by-value and copy-on-modify semantics, but at the expense of extra time and memory requirements.

  ‘MatrixExtra‘ will instead sort the indices in-place when needed, only doing deep copies when the values are to be cast to a different type, so that the object being passed to a given function or operator will always remain usable (having the same values at the same indices). However, when the indices of a given object are unsorted and get sorted in-place, if any other R object has references to the same vector of indices separately from the vector of values or vice-versa, those objects will see a new vector which is modified, potentially rendering those external objects unusable.

  If this behavior is problematic, it can be changed back to always making deep copies through ‘options("MatrixExtra.inplace_sort" = FALSE)‘.

- When doing elementwise multiplication (‘*‘) of a sparse matrix by a dense matrix or vice-versa, if the dense matrix has a ‘NA‘ or ‘NaN‘ element in a coordinate at which the sparse matrix has no entry, ‘Matrix‘ will preserve the ‘NA‘ in the resulting output, as does base R. This also applies with the propagation of 1s when using the ‘^‘ operator with zeros in the RHS, and with propagation of infinities in division by zero.

  ‘MatrixExtra‘ will instead ignore such ‘NA‘s, which makes the operation much faster. The old ‘Matrix‘ behavior can be brought back through ‘options("MatrixExtra.ignore_na" = FALSE)‘.
- When doing sparse-dense matrix multiplications, ‘Matrix‘ will run single-threaded, which is typically undesirable as modern CPUs have the capacity to perform such operations much faster by exploiting both multi-threading and SIMD instructions.

  ‘MatrixExtra‘ will by default use all the available threads in the system. The number of threads can be controlled through ‘options("MatrixExtra.nthreads" = 1L)‘.
- When calling method ‘show‘ on a sparse matrix object (for example, by typing the corresponding variable name in an R console and pressing ’Enter’, ‘Matrix‘ will take a glance at the object by printing the values in some or all of the rows and columns, typically involving a long output which, for a mid-to-large sparse matrix, will typically be larger than the available screen space and can make it very inconvenient to quickly inspect sparse objects.

  ‘MatrixExtra‘ will instead override this method with a quicker one that will only show the type, dimensions, and number of entries in a sparse object, without printing the values per-se as ‘Matrix‘ would do. Note that this only overrides the ‘show‘ method, while ‘print‘ remains the same as it was. It can be changed back to the same ‘show‘ that is provided by ‘Matrix‘ through ‘options("MatrixExtra.quick_show" = FALSE)‘.

These behaviors will change at the moment one loads ‘MatrixExtra‘ through ‘library(MatrixExtra)‘, save for the number of threads which will also be set to the maximum when calling functions as e.g. ‘MatrixExtra::<function>(<args>)‘

The package provides two short-hand functions to switch all these options simultaneously:

- ‘restore_old_matrix_behavior‘: Will change the transpose behavior to deep transposes in the same format, drop behavior to dense vectors, avoid in-place sorting of sparse indices, preserve NAs that are not in a sparse matrix in elementwise multiplication, and set the number of threads to 1.

  These all match with how the ‘Matrix‘ package behaves in such situations.
- ‘set_new_matrix_behavior‘: Will change the transpose behavior to shallow transposes in the opposite format, drop behavior to sparse vectors, sort sparse indices in-place when possible, ignore NAs that are not in a sparse matrix in elementwise multiplication, and set the number of threads to the available number of threads in the system.

  These all differ with how the ‘Matrix‘ package behaves in such situations.

Usage

```
set_new_matrix_behavior()

restore_old_matrix_behavior()
```

Value

No return value, called for side effects.

All options

List of options with the short-hand command to make them match with ‘Matrix‘.

- ‘options("MatrixExtra.fast_transpose" = FALSE)‘ : Option for behavior of ‘t(CSR)‘ and ‘t(CSC)‘.
- ‘options("MatrixExtra.drop_sparse" = FALSE)‘ : Option for behavior of ‘drop=TRUE‘ when subsetting.
- ‘options("MatrixExtra.inplace_sort" = FALSE)‘ : Option for behavior of inplace-vs-copy operations.
- ‘options("MatrixExtra.ignore_na" = FALSE)‘ : Option for behavior of ‘NA‘s in elementwise sparse-dense multipliction.
- ‘options("MatrixExtra.nthreads" = 1L)‘ : Number of parallel threads to use in sparse-dense matrix
- ‘options("MatrixExtra.quick_show" = FALSE)‘ : Option for behavior of ‘show‘ method for sparse objects.

---

operators *Mathematical operators on sparse matrices and sparse vectors*

Description

Implements some mathematical operators between sparse-sparse and sparse-dense matrices and vectors, such as ‘CSR + CSR‘, ‘CSR + COO‘, ‘CSR * vector‘, ‘CSR * dense‘, among others, which typically work natively in the storage order of the inputs without data duplication.

Usage

```
## S4 method for signature 'RsparseMatrix,sparseMatrix'
e1 * e2

## S4 method for signature 'ngRMatrix,sparseMatrix'
e1 * e2

## S4 method for signature 'lgRMatrix,sparseMatrix'
e1 * e2

## S4 method for signature 'sparseMatrix,RsparseMatrix'
e1 * e2

## S4 method for signature 'sparseMatrix,ngRMatrix'
e1 * e2

## S4 method for signature 'sparseMatrix,lgRMatrix'
e1 * e2

## S4 method for signature 'CsparseMatrix,TsparseMatrix'
```

```
e1 * e2

## S4 method for signature 'TsparseMatrix,CsparseMatrix'
e1 * e2

## S4 method for signature 'RsparseMatrix,sparseMatrix'
e1 & e2

## S4 method for signature 'ngRMatrix,sparseMatrix'
e1 & e2

## S4 method for signature 'lgRMatrix,sparseMatrix'
e1 & e2

## S4 method for signature 'sparseMatrix,RsparseMatrix'
e1 & e2

## S4 method for signature 'sparseMatrix,ngRMatrix'
e1 & e2

## S4 method for signature 'sparseMatrix,lgRMatrix'
e1 & e2

## S4 method for signature 'CsparseMatrix,TsparseMatrix'
e1 & e2

## S4 method for signature 'TsparseMatrix,CsparseMatrix'
e1 & e2

## S4 method for signature 'RsparseMatrix,matrix'
e1 * e2

## S4 method for signature 'ngRMatrix,matrix'
e1 * e2

## S4 method for signature 'lgRMatrix,matrix'
e1 * e2

## S4 method for signature 'RsparseMatrix,float32'
e1 * e2

## S4 method for signature 'ngRMatrix,float32'
e1 * e2

## S4 method for signature 'lgRMatrix,float32'
e1 * e2

## S4 method for signature 'matrix,RsparseMatrix'
```

```
e1 * e2

## S4 method for signature 'matrix,ngRMatrix'
e1 * e2

## S4 method for signature 'matrix,lgRMatrix'
e1 * e2

## S4 method for signature 'float32,RsparseMatrix'
e1 * e2

## S4 method for signature 'float32,ngRMatrix'
e1 * e2

## S4 method for signature 'float32,lgRMatrix'
e1 * e2

## S4 method for signature 'RsparseMatrix,matrix'
e1 & e2

## S4 method for signature 'ngRMatrix,matrix'
e1 & e2

## S4 method for signature 'lgRMatrix,matrix'
e1 & e2

## S4 method for signature 'matrix,RsparseMatrix'
e1 & e2

## S4 method for signature 'matrix,ngRMatrix'
e1 & e2

## S4 method for signature 'matrix,lgRMatrix'
e1 & e2

## S4 method for signature 'TsparseMatrix,matrix'
e1 * e2

## S4 method for signature 'TsparseMatrix,float32'
e1 * e2

## S4 method for signature 'ngTMatrix,matrix'
e1 * e2

## S4 method for signature 'lgTMatrix,matrix'
e1 * e2

## S4 method for signature 'ngTMatrix,float32'
```

```
e1 * e2

## S4 method for signature 'lgTMatrix,float32'
e1 * e2

## S4 method for signature 'matrix,TsparseMatrix'
e1 * e2

## S4 method for signature 'float32,TsparseMatrix'
e1 * e2

## S4 method for signature 'matrix,ngTMatrix'
e1 * e2

## S4 method for signature 'matrix,lgTMatrix'
e1 * e2

## S4 method for signature 'float32,ngTMatrix'
e1 * e2

## S4 method for signature 'float32,lgTMatrix'
e1 * e2

## S4 method for signature 'TsparseMatrix,matrix'
e1 & e2

## S4 method for signature 'ngTMatrix,matrix'
e1 & e2

## S4 method for signature 'lgTMatrix,matrix'
e1 & e2

## S4 method for signature 'matrix,TsparseMatrix'
e1 & e2

## S4 method for signature 'matrix,ngTMatrix'
e1 & e2

## S4 method for signature 'matrix,lgTMatrix'
e1 & e2

## S4 method for signature 'CsparseMatrix,matrix'
e1 * e2

## S4 method for signature 'CsparseMatrix,float32'
e1 * e2

## S4 method for signature 'matrix,CsparseMatrix'
```

```
e1 * e2

## S4 method for signature 'float32,CsparseMatrix'
e1 * e2

## S4 method for signature 'CsparseMatrix,matrix'
e1 & e2

## S4 method for signature 'CsparseMatrix,float32'
e1 & e2

## S4 method for signature 'matrix,CsparseMatrix'
e1 & e2

## S4 method for signature 'float32,CsparseMatrix'
e1 & e2

## S4 method for signature 'RsparseMatrix,sparseMatrix'
e1 + e2

## S4 method for signature 'ngRMatrix,sparseMatrix'
e1 + e2

## S4 method for signature 'lgRMatrix,sparseMatrix'
e1 + e2

## S4 method for signature 'sparseMatrix,RsparseMatrix'
e1 + e2

## S4 method for signature 'sparseMatrix,ngRMatrix'
e1 + e2

## S4 method for signature 'sparseMatrix,lgRMatrix'
e1 + e2

## S4 method for signature 'CsparseMatrix,TsparseMatrix'
e1 + e2

## S4 method for signature 'TsparseMatrix,CsparseMatrix'
e1 + e2

## S4 method for signature 'RsparseMatrix,sparseMatrix'
e1 - e2

## S4 method for signature 'ngRMatrix,sparseMatrix'
e1 - e2

## S4 method for signature 'lgRMatrix,sparseMatrix'
```

```
e1 - e2

## S4 method for signature 'sparseMatrix,RsparseMatrix'
e1 - e2

## S4 method for signature 'sparseMatrix,ngRMatrix'
e1 - e2

## S4 method for signature 'sparseMatrix,lgRMatrix'
e1 - e2

## S4 method for signature 'CsparseMatrix,TsparseMatrix'
e1 - e2

## S4 method for signature 'TsparseMatrix,CsparseMatrix'
e1 - e2

## S4 method for signature 'RsparseMatrix,sparseMatrix'
e1 | e2

## S4 method for signature 'ngRMatrix,sparseMatrix'
e1 | e2

## S4 method for signature 'lgRMatrix,sparseMatrix'
e1 | e2

## S4 method for signature 'sparseMatrix,RsparseMatrix'
e1 | e2

## S4 method for signature 'sparseMatrix,ngRMatrix'
e1 | e2

## S4 method for signature 'sparseMatrix,lgRMatrix'
e1 | e2

## S4 method for signature 'CsparseMatrix,TsparseMatrix'
e1 | e2

## S4 method for signature 'TsparseMatrix,CsparseMatrix'
e1 | e2

## S4 method for signature 'RsparseMatrix,integer'
e1 * e2

## S4 method for signature 'RsparseMatrix,numeric'
e1 * e2

## S4 method for signature 'RsparseMatrix,logical'
```

```
e1 * e2

## S4 method for signature 'integer,RsparseMatrix'
e1 * e2

## S4 method for signature 'numeric,RsparseMatrix'
e1 * e2

## S4 method for signature 'logical,RsparseMatrix'
e1 * e2

## S4 method for signature 'RsparseMatrix,integer'
e1 & e2

## S4 method for signature 'RsparseMatrix,numeric'
e1 & e2

## S4 method for signature 'RsparseMatrix,logical'
e1 & e2

## S4 method for signature 'integer,RsparseMatrix'
e1 & e2

## S4 method for signature 'numeric,RsparseMatrix'
e1 & e2

## S4 method for signature 'logical,RsparseMatrix'
e1 & e2

## S4 method for signature 'RsparseMatrix,integer'
e1 / e2

## S4 method for signature 'RsparseMatrix,numeric'
e1 / e2

## S4 method for signature 'RsparseMatrix,logical'
e1 / e2

## S4 method for signature 'RsparseMatrix,matrix'
e1 / e2

## S4 method for signature 'integer,RsparseMatrix'
e1 / e2

## S4 method for signature 'numeric,RsparseMatrix'
e1 / e2

## S4 method for signature 'logical,RsparseMatrix'
```

```
e1 / e2

## S4 method for signature 'matrix,RsparseMatrix'
e1 / e2

## S4 method for signature 'RsparseMatrix,integer'
e1 %% e2

## S4 method for signature 'RsparseMatrix,numeric'
e1 %% e2

## S4 method for signature 'RsparseMatrix,logical'
e1 %% e2

## S4 method for signature 'RsparseMatrix,matrix'
e1 %% e2

## S4 method for signature 'integer,RsparseMatrix'
e1 %% e2

## S4 method for signature 'numeric,RsparseMatrix'
e1 %% e2

## S4 method for signature 'logical,RsparseMatrix'
e1 %% e2

## S4 method for signature 'matrix,RsparseMatrix'
e1 %% e2

## S4 method for signature 'RsparseMatrix,integer'
e1 %/% e2

## S4 method for signature 'RsparseMatrix,numeric'
e1 %/% e2

## S4 method for signature 'RsparseMatrix,logical'
e1 %/% e2

## S4 method for signature 'RsparseMatrix,matrix'
e1 %/% e2

## S4 method for signature 'integer,RsparseMatrix'
e1 %/% e2

## S4 method for signature 'numeric,RsparseMatrix'
e1 %/% e2

## S4 method for signature 'logical,RsparseMatrix'
```

```
e1 %/% e2

## S4 method for signature 'matrix,RsparseMatrix'
e1 %/% e2

## S4 method for signature 'RsparseMatrix,integer'
e1 ^ e2

## S4 method for signature 'RsparseMatrix,numeric'
e1 ^ e2

## S4 method for signature 'RsparseMatrix,logical'
e1 ^ e2

## S4 method for signature 'RsparseMatrix,matrix'
e1 ^ e2

## S4 method for signature 'integer,RsparseMatrix'
e1 ^ e2

## S4 method for signature 'numeric,RsparseMatrix'
e1 ^ e2

## S4 method for signature 'logical,RsparseMatrix'
e1 ^ e2

## S4 method for signature 'matrix,RsparseMatrix'
e1 ^ e2

## S4 method for signature 'TsparseMatrix,integer'
e1 * e2

## S4 method for signature 'TsparseMatrix,numeric'
e1 * e2

## S4 method for signature 'TsparseMatrix,logical'
e1 * e2

## S4 method for signature 'integer,TsparseMatrix'
e1 * e2

## S4 method for signature 'numeric,TsparseMatrix'
e1 * e2

## S4 method for signature 'logical,TsparseMatrix'
e1 * e2

## S4 method for signature 'TsparseMatrix,integer'
```

```
e1 & e2

## S4 method for signature 'TsparseMatrix,numeric'
e1 & e2

## S4 method for signature 'TsparseMatrix,logical'
e1 & e2

## S4 method for signature 'integer,TsparseMatrix'
e1 & e2

## S4 method for signature 'numeric,TsparseMatrix'
e1 & e2

## S4 method for signature 'logical,TsparseMatrix'
e1 & e2

## S4 method for signature 'TsparseMatrix,integer'
e1 / e2

## S4 method for signature 'TsparseMatrix,numeric'
e1 / e2

## S4 method for signature 'TsparseMatrix,logical'
e1 / e2

## S4 method for signature 'TsparseMatrix,matrix'
e1 / e2

## S4 method for signature 'integer,TsparseMatrix'
e1 / e2

## S4 method for signature 'numeric,TsparseMatrix'
e1 / e2

## S4 method for signature 'logical,TsparseMatrix'
e1 / e2

## S4 method for signature 'matrix,TsparseMatrix'
e1 / e2

## S4 method for signature 'TsparseMatrix,integer'
e1 %% e2

## S4 method for signature 'TsparseMatrix,numeric'
e1 %% e2

## S4 method for signature 'TsparseMatrix,logical'
```

```
e1 %% e2

## S4 method for signature 'TsparseMatrix,matrix'
e1 %% e2

## S4 method for signature 'integer,TsparseMatrix'
e1 %% e2

## S4 method for signature 'numeric,TsparseMatrix'
e1 %% e2

## S4 method for signature 'logical,TsparseMatrix'
e1 %% e2

## S4 method for signature 'matrix,TsparseMatrix'
e1 %% e2

## S4 method for signature 'TsparseMatrix,integer'
e1 %/% e2

## S4 method for signature 'TsparseMatrix,numeric'
e1 %/% e2

## S4 method for signature 'TsparseMatrix,logical'
e1 %/% e2

## S4 method for signature 'TsparseMatrix,matrix'
e1 %/% e2

## S4 method for signature 'integer,TsparseMatrix'
e1 %/% e2

## S4 method for signature 'numeric,TsparseMatrix'
e1 %/% e2

## S4 method for signature 'logical,TsparseMatrix'
e1 %/% e2

## S4 method for signature 'matrix,TsparseMatrix'
e1 %/% e2

## S4 method for signature 'TsparseMatrix,integer'
e1 ^ e2

## S4 method for signature 'TsparseMatrix,numeric'
e1 ^ e2

## S4 method for signature 'TsparseMatrix,logical'
```

```
e1 ^ e2

## S4 method for signature 'TsparseMatrix,matrix'
e1 ^ e2

## S4 method for signature 'integer,TsparseMatrix'
e1 ^ e2

## S4 method for signature 'numeric,TsparseMatrix'
e1 ^ e2

## S4 method for signature 'logical,TsparseMatrix'
e1 ^ e2

## S4 method for signature 'matrix,TsparseMatrix'
e1 ^ e2

## S4 method for signature 'RsparseMatrix,sparseVector'
e1 * e2

## S4 method for signature 'sparseVector,RsparseMatrix'
e1 * e2

## S4 method for signature 'matrix,sparseVector'
e1 * e2

## S4 method for signature 'sparseVector,matrix'
e1 * e2

## S4 method for signature 'float32,sparseVector'
e1 * e2

## S4 method for signature 'sparseVector,float32'
e1 * e2
```

Arguments

e1	A sparse or dense matrix or vector/
e2	Another sparse or dense matrix or vector.

Details

By default, when doing elementwise multiplication (‘*‘) between a sparse and a dense matrix or vice-versa, if the dense matrix has missing values (‘NA‘ / ‘NaN‘) at some coordinate in which the sparse matrix has no present entry, the resulting output will not have an entry there either, which differs from the behavior of ‘Matrix‘ and base R, but makes the operation much faster. The same applies to division by zero and exponentiation to zero.

If such missing values (or infinites and ones) are to be preserved, this behavior can be changed

through the package options (i.e. ‘options("MatrixExtra.ignore_na" = FALSE)‘ - see MatrixExtra-options).

The indices of the matrices might be sorted in-place for some operations (see sort_sparse_indices).

Value

A CSR or COO matrix depending on the input type and operation. Some operations (blocked by default) will produce dense matrices as outputs.

Examples

```
library(Matrix)
library(MatrixExtra)
set.seed(1)
X <- rsparsematrix(4, 3, .5, repr="R")
options("MatrixExtra.quick_show" = FALSE)
X + X
X * X
X * as.coo.matrix(X)
X * 2
X * 1:4
X ^ 2
X ^ (1:4)

### Beware
set_new_matrix_behavior()
print(suppressWarnings(X / 0))
restore_old_matrix_behavior()
print(suppressWarnings(X / 0))
```

---

rbind2-method *Concatenate sparse matrices/vectors by rows*

Description

‘rbind2‘ method for the sparse matrix and sparse vector classes from ‘Matrix‘, taking the most efficient route for the concatenation according to the input types.

Usage

```
## S4 method for signature 'RsparseMatrix,RsparseMatrix'
rbind2(x, y)

## S4 method for signature 'sparseVector,RsparseMatrix'
rbind2(x, y)

## S4 method for signature 'RsparseMatrix,sparseVector'
rbind2(x, y)
```

```
## S4 method for signature 'sparseVector,sparseVector'
rbind2(x, y)

## S4 method for signature 'CsparseMatrix,CsparseMatrix'
rbind2(x, y)

## S4 method for signature 'sparseVector,CsparseMatrix'
rbind2(x, y)

## S4 method for signature 'CsparseMatrix,sparseVector'
rbind2(x, y)

## S4 method for signature 'CsparseMatrix,CsparseMatrix'
rbind2(x, y)

## S4 method for signature 'RsparseMatrix,CsparseMatrix'
rbind2(x, y)

## S4 method for signature 'CsparseMatrix,RsparseMatrix'
rbind2(x, y)

## S4 method for signature 'RsparseMatrix,TsparseMatrix'
rbind2(x, y)

## S4 method for signature 'TsparseMatrix,RsparseMatrix'
rbind2(x, y)

## S4 method for signature 'RsparseMatrix,numeric'
rbind2(x, y)

## S4 method for signature 'RsparseMatrix,integer'
rbind2(x, y)

## S4 method for signature 'RsparseMatrix,logical'
rbind2(x, y)

## S4 method for signature 'numeric,RsparseMatrix'
rbind2(x, y)

## S4 method for signature 'integer,RsparseMatrix'
rbind2(x, y)

## S4 method for signature 'logical,RsparseMatrix'
rbind2(x, y)

## S4 method for signature 'TsparseMatrix,TsparseMatrix'
rbind2(x, y)
```

```
## S4 method for signature 'TsparseMatrix,sparseVector'
rbind2(x, y)

## S4 method for signature 'sparseVector,TsparseMatrix'
rbind2(x, y)

## S4 method for signature 'TsparseMatrix,CsparseMatrix'
rbind2(x, y)

## S4 method for signature 'CsparseMatrix,TsparseMatrix'
rbind2(x, y)
```

Arguments

x	First matrix to concatenate.
y	Second matrix to concatenate.

Value

A sparse matrix, usually in CSR format but some combinations might return COO or CSC.

Examples

```
library(Matrix)
library(MatrixExtra)
set.seed(1)
X <- rsparsematrix(3, 4, .3)
X <- as(X, "RsparseMatrix")
inherits(rbind2(X, X), "RsparseMatrix")
inherits(rbind(X, X, as.csc.matrix(X), X), "RsparseMatrix")
inherits(rbind2(as.coo.matrix(X), as.coo.matrix(X)), "TsparseMatrix")
inherits(rbind2(as.csc.matrix(X), as.csc.matrix(X)), "CsparseMatrix")
```

---

rbind_csr *Concatenate inputs by rows into a CSR matrix*

Description

Concatenate two or more matrices and/or vectors by rows, giving a CSR matrix as result.

This is aimed at concatenating several CSR matrices or sparse vectors at a time, as it will be faster than calling ‘rbind‘ which will only concatenate one at a time, resulting in unnecessary allocations.

Usage

```
rbind_csr(...)
```

Arguments

...	Inputs to concatenate. The function is aimed at CSR matrices ('dgRMatrix', 'ngRMatrix', 'lgRMatrix') and sparse vectors ('sparseVector'). It will work with other classes (such as 'dgCMatrix') but will not be as efficient.

Details

This function will not preserve the column names, if any were present.

Value

A CSR matrix (class 'dgRMatrix', 'lgRMatrix', or 'ngRMatrix' depending on the inputs) with the inputs concatenated by rows.

See Also

rbind2-method

Examples

```
library(Matrix)
library(MatrixExtra)
options("MatrixExtra.quick_show" = FALSE)
v <- as(1:10, "sparseVector")
rbind_csr(v, v, v)

X <- matrix(1:20, nrow=2)
rbind_csr(X, v)
```

---

remove_sparse_zeros *Remove Zeros from a Sparse Matrix or Sparse Vector*

Description

Removes the entries in a sparse matrix or sparse vector which have a value of zero but nevertheless are still among the object's values, in any case there are any. Can also remove missing values if desired.

Usage

```
remove_sparse_zeros(X, na.rm = FALSE)
```

Arguments

X	A sparse matrix (COO, CSR, CSC) or sparse vector (any type) from the 'Matrix' package, whose values will be removed (left as non-present in the sparse representation) if they are zeros.
na.rm	Whether to also remove missing values ('NA' / 'NaN') from 'X'.

Value

The same matrix / vector X with its zeros removed from the sparse representation.

---

show *Quick Glance at Sparse Objects*

Description

Shows some basic information about a sparse matrix or sparse vector object, without printing a subset of its entries as ‘Matrix‘ would do.

Note that this package will by default override the ‘show‘ methods of sparse objects, but not the ‘print‘ methods - for example, if one defines a variable ’X’ containing a sparse matrix, and then types ’X’ in the console, that calls the ‘show‘ method, but one can still print it by calling ’print(X)’.

In order to restore the ‘show‘ method provided by ‘Matrix‘, call ‘options("MatrixExtra.quick_show" = FALSE)‘.

Usage

```
## S4 method for signature 'sparseMatrix'
show(object)

## S4 method for signature 'sparseVector'
show(object)

## S4 method for signature 'sparseVector'
print(x)
```

Arguments

object	A sparse matrix or sparse vector.
x	A sparse vector (same method as in matrix, readded here to avoid naming conflicts).

Value

The same object that was passed as input, as invisible.

Examples

```
library(Matrix)
library(MatrixExtra)

set.seed(1)
X <- Matrix::rsparsematrix(5, 5, .2)
set_new_matrix_behavior()
show(X)
print(X)
```

```
X

restore_old_matrix_behavior()
show(X)
print(X)
X
```

---

slice *Sparse Matrices Slicing*

Description

Natively slice CSR/CSC/COO matrices without changing the storage order.

Usage

```
## S4 method for signature 'RsparseMatrix,index,index,logical'
x[i, j, drop]

## S4 method for signature 'RsparseMatrix,missing,index,logical'
x[i, j, drop]

## S4 method for signature 'RsparseMatrix,index,missing,logical'
x[i, j, drop]

## S4 method for signature 'RsparseMatrix,missing,missing,logical'
x[i, j, drop]

## S4 method for signature 'RsparseMatrix,index,index,missing'
x[i, j, drop]

## S4 method for signature 'RsparseMatrix,missing,index,missing'
x[i, j, drop]

## S4 method for signature 'RsparseMatrix,index,missing,missing'
x[i, j, drop]

## S4 method for signature 'RsparseMatrix,missing,missing,missing'
x[i, j, drop]

## S4 method for signature 'ANY,nsparseVector,nsparseVector,logical'
x[i, j, drop]

## S4 method for signature 'ANY,missing,nsparseVector,logical'
x[i, j, drop]

## S4 method for signature 'ANY,nsparseVector,missing,logical'
```

```
x[i, j, drop]

## S4 method for signature 'ANY,index,nsparseVector,logical'
x[i, j, drop]

## S4 method for signature 'ANY,nsparseVector,index,logical'
x[i, j, drop]

## S4 method for signature 'ANY,nsparseVector,nsparseVector,missing'
x[i, j, drop]

## S4 method for signature 'ANY,missing,nsparseVector,missing'
x[i, j, drop]

## S4 method for signature 'ANY,nsparseVector,missing,missing'
x[i, j, drop]

## S4 method for signature 'ANY,index,nsparseVector,missing'
x[i, j, drop]

## S4 method for signature 'ANY,nsparseVector,index,missing'
x[i, j, drop]

## S4 method for signature 'ANY,lsparseVector,lsparseVector,logical'
x[i, j, drop]

## S4 method for signature 'ANY,missing,lsparseVector,logical'
x[i, j, drop]

## S4 method for signature 'ANY,lsparseVector,missing,logical'
x[i, j, drop]

## S4 method for signature 'ANY,index,lsparseVector,logical'
x[i, j, drop]

## S4 method for signature 'ANY,lsparseVector,index,logical'
x[i, j, drop]

## S4 method for signature 'ANY,lsparseVector,lsparseVector,missing'
x[i, j, drop]

## S4 method for signature 'ANY,missing,lsparseVector,missing'
x[i, j, drop]

## S4 method for signature 'ANY,lsparseVector,missing,missing'
x[i, j, drop]

## S4 method for signature 'ANY,index,lsparseVector,missing'
```

```
x[i, j, drop]

## S4 method for signature 'ANY,lsparseVector,index,missing'
x[i, j, drop]

## S4 method for signature 'CsparseMatrix,index,index,logical'
x[i, j, drop]

## S4 method for signature 'CsparseMatrix,missing,index,logical'
x[i, j, drop]

## S4 method for signature 'CsparseMatrix,index,missing,logical'
x[i, j, drop]

## S4 method for signature 'CsparseMatrix,missing,missing,logical'
x[i, j, drop]

## S4 method for signature 'CsparseMatrix,index,index,missing'
x[i, j, drop]

## S4 method for signature 'CsparseMatrix,missing,index,missing'
x[i, j, drop]

## S4 method for signature 'CsparseMatrix,index,missing,missing'
x[i, j, drop]

## S4 method for signature 'CsparseMatrix,missing,missing,missing'
x[i, j, drop]

## S4 method for signature 'TsparseMatrix,index,index,logical'
x[i, j, drop]

## S4 method for signature 'TsparseMatrix,missing,index,logical'
x[i, j, drop]

## S4 method for signature 'TsparseMatrix,index,missing,logical'
x[i, j, drop]

## S4 method for signature 'TsparseMatrix,missing,missing,logical'
x[i, j, drop]

## S4 method for signature 'TsparseMatrix,index,index,missing'
x[i, j, drop]

## S4 method for signature 'TsparseMatrix,missing,index,missing'
x[i, j, drop]

## S4 method for signature 'TsparseMatrix,index,missing,missing'
```

```
x[i, j, drop]

## S4 method for signature 'TsparseMatrix,missing,missing,missing'
x[i, j, drop]
```

Arguments

x	A sparse matrix to subset, in any format.
i	row indices to subset.
j	column indices to subset.
drop	whether to simplify 1d matrix to a vector

Details

Important: When slicing sparse matrices with ‘drop=TRUE‘ (the default), ‘Matrix‘ will drop 1-d matrices to **dense** dense vectors, whereas this package allows dropping them to either dense or **sparse** vectors, the latter of which is more efficient and is the default option.

The ‘drop‘ behavior can be changed back to dense vectors like ‘Matrix‘ does, through restore_old_matrix_behavior or through the package options (e.g. ‘options("MatrixExtra.drop_sparse" = FALSE)‘ - see MatrixExtra-options).

Note: Trying to slice a sparse matrix without supplying any parameter for the second axis (e.g. ‘X[1:10]‘) will **select whole rows** (as if it were ‘X[1:10,]‘) instead of selecting entries as if the input were a flattened array (which is what ‘Matrix‘ and base R do).

This package will override the subsetting methods from ‘Matrix‘ for all sparse matrix types. It is usually much faster for all three storage orders (especially CSR) but in some situations could end up being slightly slower. Be aware that, in the case of COO matrices (a.k.a. "TsparseMatrix"), the resulting object will **not** have sorted indices, which ‘Matrix‘ will oftentimes do in addition to subsetting, at a large speed penalty.

In general, it’s much faster to select rows when the input is a CSR matrix ("RsparseMatrix"), and much faster to select columns when the input is a CSC matrix ("CsparseMatrix"). Slicing COO matrices is typically not efficient, but could end up being faster when the slice involves random rows and random columns with repeated entries.

Value

A sparse matrix with the same storage order and dtype as ‘x‘.

Examples

```
library(Matrix)
library(MatrixExtra)
m <- rsparsematrix(20, 20, 0.1, repr="R")
inherits(m[1:2, ], "RsparseMatrix")
inherits(m[1:2, 3:4], "RsparseMatrix")
inherits(as.coo.matrix(m)[1:2, 3:4], "TsparseMatrix")
inherits(as.csc.matrix(m)[1:2, 3:4], "CsparseMatrix")

### New: slice with a sparse vector
```

```
m[as(c(TRUE,FALSE), "sparseVector"), ]

### Important!!!
### This differs from Matrix
set_new_matrix_behavior()
inherits(m[1,,drop=TRUE], "sparseVector")

### To bring back the old behavior:
restore_old_matrix_behavior()
inherits(m[1,,drop=TRUE], "numeric")
```

`sort_sparse_indices` *Sort the indices of a sparse matrix or sparse vector*

Description

Will sort the indices of a sparse matrix or sparse vector.

In general, the indices of sparse CSR and CSC matrices are always meant to be sorted, and it should be rare to have a matrix with unsorted indices when the matrix is the output of some built-in operation from either 'Matrix' or 'MatrixExtra', but when the matrices are constructed manually this function can come in handy.

Important: the input matrix will be modified in-place, unless passing 'copy=TRUE'.

Usage

```
sort_sparse_indices(X, copy = FALSE, byrow = TRUE)
```

Arguments

`X`	A sparse matrix in CSR, CSC, or COO format; or a sparse vector (from the 'Matrix' package.)
`copy`	Whether to make a deep copy of the indices and the values before sorting them, so that the in-place modifications will not affect any potential external references to the same arrays.
`byrow`	When passing a COO matrix ("TsparseMatrix"), whether to sort it with rows as the major axis, which differs from 'Matrix' that usually sorts them with columns as the major axis.

Value

The same input 'X' with its indices sorted, as invisible (no auto print). Note that the input is itself modified, so there is no need to reassign it.

t_shallow *Transpose a sparse matrix by changing its format*

Description

Transposes a sparse matrix in CSC (a.k.a. "CsparseMatrix") or CSR (a.k.a. "RsparseMatrix") formats by converting it to the opposite format (i.e. CSC -> CSR, CSR -> CSC).

This implies only a shallow copy (i.e. it's much faster), as the only necessary thing to make such transpose operation is to swap the number of rows and columns and change the class of the object (all data remains the same), avoiding any deep copying and format conversion as when e.g. creating a CSC transpose of a CSC matrix.

If the input is neither a CSR not CSC matrix, it will just call the generic 't()' method.

Also provided is a function 't_deep' which outputs a transpose with the same storage order.

Usage

```
t_shallow(x)

t_deep(x)

## S4 method for signature 'RsparseMatrix'
t(x)

## S4 method for signature 'CsparseMatrix'
t(x)

## S4 method for signature 'TsparseMatrix'
t(x)

## S4 method for signature 'dgCMatrix'
t(x)

## S4 method for signature 'ngCMatrix'
t(x)

## S4 method for signature 'lgCMatrix'
t(x)

## S4 method for signature 'dtCMatrix'
t(x)

## S4 method for signature 'ntCMatrix'
t(x)

## S4 method for signature 'ltCMatrix'
t(x)
```

```
## S4 method for signature 'dsCMatrix'
t(x)

## S4 method for signature 'nsCMatrix'
t(x)

## S4 method for signature 'lsCMatrix'
t(x)

## S4 method for signature 'sparseVector'
t(x)
```

Arguments

x	A sparse matrix. If ‘x‘ is of a different type, will just invoke its generic ‘t()‘ method.

Details

Important: When loading this package (‘library(MatrixExtra)‘), it will change the behavior of ‘t(sparseMatrix)‘ towards calling ‘t_shallow‘.

This makes it more efficient, but has the potential of breaking existing code in other packages, particularly in the ‘Matrix‘ package itself when calling some arbitrary function or method which would internally transpose a CSC matrix and rely on the assumption that its output is also CSC.

This behavior can be changed through restore_old_matrix_behavior or the package options (e.g. ‘options("MatrixExtra.fast_transpose" = FALSE)‘ - ee MatrixExtra-options) to have ‘t_deep‘ as the default, just like in ‘Matrix‘.

Additionally, under the new behavior (‘t_shallow‘ as the default for ‘t‘), transposing a ‘sparseVector‘ object will yield a CSR matrix ("RsparseMatrix"), which differs from ‘Matrix‘ that would yield a COO matrix ("TsparseMatrix").

Value

The transpose of ‘x‘ (rows become columns and columns become rows), but in the opposite format (CSC -> CSR, CSR -> CSC); or the same format if calling ‘t_deep‘.

Examples

```
library(Matrix)
library(MatrixExtra)
set.seed(1)
X <- rsparsematrix(3, 4, .5, repr="C")
inherits(X, "CsparseMatrix")
Xtrans <- t_shallow(X)
inherits(Xtrans, "RsparseMatrix")
nrow(X) == ncol(Xtrans)
ncol(X) == nrow(Xtrans)
```

```
Xorig <- t_shallow(Xtrans)
inherits(Xorig, "CsparseMatrix")
inherits(t_deep(Xtrans), "RsparseMatrix")

### Important!!!
### This package makes 't_shallow' the default
set_new_matrix_behavior()
inherits(X, "CsparseMatrix")
inherits(t(X), "RsparseMatrix")

### Can be changed back to 't_deep' like this:
restore_old_matrix_behavior()
inherits(t(X), "CsparseMatrix")
```

Index