

# Package ‘BumpyMatrix’

July 15, 2025

**Version** 1.17.0

**Date** 2021-07-03

**Title** Bumpy Matrix of Non-Scalar Objects

**Description** Implements the BumpyMatrix class and several subclasses for holding non-scalar objects in each entry of the matrix. This is akin to a ragged array but the raggedness is in the third dimension, much like a bumpy surface - hence the name. Of particular interest is the BumpyDataFrameMatrix, where each entry is a Bioconductor data frame. This allows us to naturally represent multivariate data in a format that is compatible with two-dimensional containers like the SummarizedExperiment and MultiAssayExperiment objects.

**License** MIT + file LICENSE

**Imports** utils, methods, Matrix, S4Vectors, IRanges

**Suggests** BiocStyle, knitr, rmarkdown, testthat

**biocViews** Software, Infrastructure, DataRepresentation

**VignetteBuilder** knitr

**RoxygenNote** 7.1.1

**URL** <https://bioconductor.org/packages/BumpyMatrix>

**BugReports** <https://github.com/LTLA/BumpyMatrix/issues>

**git\_url** <https://git.bioconductor.org/packages/BumpyMatrix>

**git\_branch** devel

**git\_last\_commit** 4c445a7

**git\_last\_commit\_date** 2025-04-15

**Repository** Bioconductor 3.22

**Date/Publication** 2025-07-15

**Author** Aaron Lun [aut, cre],  
Genentech, Inc. [cph]

**Maintainer** Aaron Lun <[infinite.monkeys.with.keyboards@gmail.com](mailto:infinite.monkeys.with.keyboards@gmail.com)>

## Contents

BumpyAtomicMatrix . . . . .	2
BumpyDataFrameMatrix . . . . .	3
BumpyMatrix . . . . .	4
splitAsBumpyMatrix . . . . .	7
unsplitAsDataFrame . . . . .	8

<b>Index</b>	<b>10</b>
--------------	-----------

---

BumpyAtomicMatrix	<i>The BumpyAtomicMatrix subclass</i>
-------------------	---------------------------------------

---

## Description

A subclass of the [BumpyMatrix](#) where each entry is an atomic vector. One subclass is provided for each of the most common types.

## Details

In the following code snippets, `x` is a `BumpyDataFrameMatrix`.

Binary and unary operations are implemented by specializing [Ops](#), [Math](#) and related group generics, and will usually return a new `BumpyAtomicMatrix` of the appropriate type. The exception is for [Summary](#) methods like `max` and `min`; these return an ordinary matrix where each entry contains a scalar value for the corresponding entry of `x`. Furthermore, `range` will return a 3-dimensional array containing the minimum and maximum for each entry of `x`.

Common mathematical operations are implemented that apply to each entry of the `BumpyAtomicMatrix`:

- `mean`, `sd`, `median`, `mad`, `var` and `IQR` take a single `BumpyAtomicMatrix` and return an ordinary double-precision matrix of the same dimensions containing the computed statistic for each entry of the input. This is possible as all operations are guaranteed to produce a scalar.
- `quantile` takes a single `BumpyAtomicMatrix` as input and return a 3-dimensional array. The first dimension contains the requested quantiles, the second dimension corresponds to the rows of `x` and the third dimension corresponds to the columns of `x`.
- `which.max` and `which.min` take a single `BumpyAtomicMatrix` and return an ordinary integer matrix of the same dimensions containing the index for the min/max value per entry. (This is set to `NA` if the entry of the input has length zero.)
- `pmin`, `pmax`, `pmin.int` and `pmax.int` take multiple `BumpyAtomicMatrix` objects of the same dimensions, and return a `BumpyAtomicMatrix` containing the result of running the same function across corresponding entries of the input objects.
- `cor`, `cov` and (optionally) `var` take two `BumpyAtomicMatrix` objects of the same dimensions, and return an ordinary matrix containing the computed statistic for the corresponding entries of the inputs. This is possible as all operations are guaranteed to produce a scalar.

Additionally, common operations are implemented that apply to each entry of the `BumpyCharacterMatrix` and return a `BumpyAtomicMatrix` of the same dimensions and an appropriate type. This includes `tolower`, `toupper`, `substr`, `substring`, `sub`, `gsub`, `grep1`, `grep`, `nchar`, `chartr`, `startsWith` and `endsWith`. We also implement `unstrsplit`, which returns an ordinary matrix of the same dimensions containing the unsplit strings.

All methods implemented for the [BumpyMatrix](#) parent class are available here.

## Examples

```
# Mocking up a BumpyNumericList:
library(IRanges)
x <- splitAsList(runif(1000), factor(sample(50, 1000, replace=TRUE), 1:50))

# Creating a BumpyNumericMatrix:
mat <- BumpyMatrix(x, c(10, 5))
mat[,1]

# Arithmetic operations:
(mat * 2)[,1]
(mat + mat * 5)[,1]

# Logical operations:
(mat < 0.5)[,1]
(mat > 0.5 & mat < 1)[,1]
(mat == mat)[,1]

# More statistics:
max(mat)
min(mat)
mean(mat)
sd(mat)
median(mat)

# Handling character vectors:
x <- splitAsList(sample(LETTERS, 100, replace=TRUE),
  factor(sample(20, 100, replace=TRUE), 1:20))
cmat <- BumpyMatrix(x, c(5, 4))
cmat[,1]

tolower(cmat[,1])
grepl("A|E|I|O|U", cmat)[,1]
sub("A|E|I|O|U", "vowel", cmat)[,1]
```

---

BumpyDataFrameMatrix    *The BumpyDataFrameMatrix class*

---

## Description

The BumpyDataFrameMatrix provides a two-dimensional object where each entry is a [DataFrame](#). This is useful for storing data that has a variable number of observations per sample/feature combination, e.g., for inclusion as another assay in a SummarizedExperiment object.

## Details

In the following code snippets, `x` is a BumpyDataFrameMatrix.

`commonColnames(x)` will return a character vector with the names of the available commonColnames. This can be modified with `commonColnames(x) <- value`.

`x[i, j, k, ..., .dropk=drop, drop=TRUE]` will subset the BumpyDataFrameMatrix:

- If `k` is not specified, this will either produce another `BumpyDataFrameMatrix` corresponding to the specified submatrix, or a `CompressedSplitDataFrameList` containing the entries of interest if `drop=TRUE`.
- If `k` is specified, it should contain the names or indices of the columns of the underlying `DataFrame` to retain. For multiple fields or with `.dropk=FALSE`, a new `BumpyDataFrameMatrix` is returned with the specified columns in the `DataFrame`.
- If `k` only specifies a single column and `.dropk=TRUE`, a `BumpyMatrix` (or `CompressedList`, if `drop=TRUE`) corresponding to the type of the field is returned.

`x[i, j, k, ...] <- value` will modify `x` by replacing the specified values with those in the `BumpyMatrix` value of the same dimensions. If `k` is not specified, `value` should be a `BumpyDataFrameMatrix` with the same fields as `x`. If `k` is specified, `value` should be a `BumpyDataFrameMatrix` with the specified fields. If `k` contains a single field, `value` can also be a `BumpyAtomicMatrix` containing the values to use in that field.

All methods described for the `BumpyMatrix` parent class are available.

### Author(s)

Aaron Lun

### Examples

```
library(S4Vectors)
df <- DataFrame(x=runif(100), y=runif(100))
f <- factor(sample(letters[1:20], nrow(df), replace=TRUE), letters[1:20])
out <- split(df, f)

# Making our BumpyDataFrameMatrix.
mat <- BumpyMatrix(out, c(5, 4))
mat[,1]
mat[1,]

# Subsetting capabilities.
xmat <- mat[, "x"]
ymat <- mat[, "y"]
filtered <- mat[xmat > 0.5 & ymat > 0.5]
filtered[,1]

# Subset replacement works as expected.
mat2 <- mat
mat2[, "x"] <- mat2[, "x"] * 2
mat2[,1]
```

---

BumpyMatrix

*The BumpyMatrix class*

---

### Description

The `BumpyMatrix` provides a two-dimensional object where each entry is a `Vector` object. This is useful for storing data that has a variable number of observations per sample/feature combination, e.g., for inclusion as another assay in a `SummarizedExperiment` object.

## Constructor

`BumpyMatrix(x, dims, dimnames=list(NULL, NULL), proxy=NULL, reorder=TRUE)` will produce a `BumpyMatrix` object, given:

- `x`, a [CompressedList](#) object containing one or more [DFrames](#) or atomic vectors.
- `dim`, an integer vector of length 2 specifying the dimensions of the returned object.
- `dimnames`, a list of length 2 containing the row and column names.
- `proxy`, an integer or numeric matrix-like object specifying the location of each entry of `x` in the output matrix.
- `reorder`, a logical scalar indicating whether `proxy` (if specified) should be reordered.

The type of the returned `BumpyMatrix` object is determined from the type of `x`.

If `proxy=NULL`, `x` should have length equal to the product of `dim`. The entries of the returned `BumpyMatrix` are filled with `x` in a column-major manner.

If `proxy` is specified, it should contain indices in `1:length(x)` with all other entries filled with zeros. If `reorder=FALSE`, all non-zero values should be in increasing order when encountered in column-major format; otherwise, the indices are resorted to enforce this expectation. Note that `dims` and `dimnames` are ignored.

If `x` is missing, a [BumpyIntegerMatrix](#) is returned with zero rows and columns. If `dim` is also specified, a `BumpyIntegerMatrix` with the specified number of rows and columns is returned, where each entry is an empty integer vector.

## Basic matrix methods

In the following code snippets, `x` is an instance of a `BumpyMatrix` subclass.

`dim(x)` will yield a length-2 integer vector containing the number of rows and columns in `x`.  
`length(x)` will yield the product of the number of columns and rows.

`dimnames(x)` will yield a list of two character vectors with the row and column names of `x`. Either or both elements of the list may be `NULL` if no names are present.

`x[i, j, ..., drop=TRUE]` will yield the specified submatrix of the same type as `x`, given integer, character or logical subsetting vectors in `i` and `j`. If the resulting submatrix has any dimension of length 1 and `drop=TRUE`, a [CompressedList](#) of the appropriate type is instead returned.

`x[i, j] <- value` will replace the specified entries in `x` with the values in another `BumpyMatrix` value. It is expected that `value` is of the same subclass as `x`. `value` can also be a [CompressedList](#) of the same class as `undim(x)`, in which case it is recycled to fill the specified entries.

`t(x)` will transpose the `BumpyMatrix`, returning an object of the same type.

`rbind(..., deparse.level=1)` and `cbind(..., deparse.level=1)` will combine all `BumpyMatrix` objects in `...`, yielding a single `BumpyMatrix` object containing all the rows and columns, respectively. All objects should have the same number of columns (for `rbind`) or rows (for `cbind`).

## Subsetting by another BumpyMatrix

Given a `BumpyMatrix` `x` and an appropriate `BumpyMatrix` `i`, `x[i]` will return another `BumpyMatrix` where each entry of `x` is subsetting by the corresponding entry of `i`. This usually requires `i` to be a `BumpyIntegerMatrix` or a `BumpyLogicalMatrix`, though it is also possible to use a `BumpyCharacterMatrix` if each entry of `x` is named.

### Special CompressedList methods

`undim(x)` will return the underlying [CompressedList](#) object.

`redim(flesh, skeleton)` will create a `BumpyMatrix` object, given a `CompressedList` `flesh` and an existing `BumpyMatrix` object `skeleton`. `flesh` is assumed to be of the same length as `undim(skeleton)` where each entry in the former replaces the corresponding entry in the latter. The class of the output is determined based on the class of `flesh`. This method is analogous to the [relist](#) function for lists.

`unlist(x, ...)` will return the underlying [Vector](#) used to create the [CompressedList](#) object. This is the same as `unlist(undim(x), ...)`.

### Other methods

`lengths(x)` will return a numeric matrix-like object with the same dimensions and `dimnames` as `x`, where each entry contains the length of the corresponding entry in `x`. The output class can be anything used in the proxy of the constructor, e.g., a sparse matrix from the **Matrix** package.

### Author(s)

Aaron Lun

### Examples

```
# Mocking up a BumpyNumericList:
library(IRanges)
x <- NumericList(split(runif(1000), factor(sample(50, 1000, replace=TRUE), 1:50)))
length(x)

# Creating a BumpyNumericMatrix:
mat <- BumpyMatrix(x, c(10, 5))
mat

# Standard subsetting works correctly:
mat[1:10,1:2]
mat[,1]
mat[1,]

# Subsetting by another BumpyMatrix.
is.big <- x > 0.9
i <- BumpyMatrix(is.big, dim(mat))
out <- mat[i]
out # same dimensions as mat...
out[,1] # but the entries are subsetted.
out[1,]

# Subset replacement works correctly:
mat[,2]
alt <- mat
alt[,2] <- mat[,1,drop=FALSE]
alt[,2]

# Combining works correctly:
rbind(mat, mat)
cbind(mat, mat)

# Transposition works correctly:
```

```
mat[1,2]
tmat <- t(mat)
tmat
tmat[1,2]

# Get the underlying objects:
undim(mat)
summary(unlist(mat))
```

---

splitAsBumpyMatrix	<i>Split to a BumpyMatrix</i>
--------------------	-------------------------------

---

## Description

Split a vector or [Vector](#) into a BumpyMatrix based on row/column factors. This facilitates the construction of a BumpyMatrix from vector-like objects.

## Usage

```
splitAsBumpyMatrix(x, row, column, sparse = FALSE)
```

## Arguments

x	A vector or <a href="#">Vector</a> object, most typically a <a href="#">DFrame</a> .
row	An object coercible into a factor, of length equal to x. This defines the row index for each element of x.
column	An object coercible into a factor, of length equal to x. This defines the column index for each element of x.
sparse	Logical scalar indicating whether a sparse representation should be used.

## Value

A [BumpyMatrix](#) of the appropriate type, with number of rows and columns equal to the number of levels in row and column respectively. Each entry of the matrix contains all elements of x with the corresponding indices in row and column.

## Author(s)

Aaron Lun

## See Also

[BumpyMatrix](#), if a [CompressedList](#) has already been constructed.

[unsplitAsDataFrame](#), which reverses this operation to recover a long-format DataFrame.

[splitAsList](#), which inspired this function.

## Examples

```
mat <- splitAsBumpyMatrix(runif(1000),
  row=sample(LETTERS, 1000, replace=TRUE),
  column=sample(10, 1000, replace=TRUE)
)
mat
mat[,1]
mat[1,]

# Or with a sparse representation.
mat <- splitAsBumpyMatrix(runif(10),
  row=sample(LETTERS, 10, replace=TRUE),
  column=sample(10, 10, replace=TRUE)
)
mat
mat[,1]
mat[1,]
```

---

unsplitAsDataFrame	<i>Unsplit a BumpyMatrix</i>
--------------------	------------------------------

---

## Description

Unsplit a BumpyMatrix into a [DataFrame](#), adding back the row and column names as separate columns. This is equivalent to converting the BumpyMatrix into a “long” format.

## Usage

```
unsplitAsDataFrame(
  x,
  row.names = TRUE,
  column.names = TRUE,
  row.field = "row",
  column.field = "column",
  value.field = "value"
)
```

## Arguments

x	A BumpyMatrix object.
row.names, column.names	Logical scalar indicating whether the row or column names of x should be reported in the output.
row.field, column.field	String indicating the field in the output DataFrame to store the row or column names.
value.field	String specifying the field in the output DataFrame to store BumpyAtomicMatrix values.



## Details

Denote the output of this function as *y*. Given a `BumpyAtomicMatrix` *x*, we would expect to be able to recover *x* by calling `splitAsBumpyMatrix(y$value, y$row, y$column)`.

The `row.field`, `column.field` and `value.field` arguments can be used to alter the column names of the output `DataFrame`. This can be helpful to avoid, e.g., conflicts with columns of the same name in a `BumpyDataFrameMatrix` *x*.

If no row/column names are present in *x* (or `row.names` or `column.names` is `FALSE`), the row and column columns instead hold integer indices specifying the matrix row/column of each `DataFrame` row.

## Value

A `DataFrame` object containing the data in *x*. This has additional row and column columns containing the row/column names for each `DataFrame` row.

If *x* is a `BumpyAtomicMatrix`, the output `DataFrame` contains a `value` column that holds `unlist(x)`. Otherwise, if *x* is a `BumpyDataFrameMatrix`, the `DataFrame` contains the columns in `unlist(x)`.

## Author(s)

Aaron Lun

## See Also

`splitAsBumpyMatrix`, to do the split in the first place.

## Examples

```
mat <- splitAsBumpyMatrix(runif(1000),
  row=sample(LETTERS, 1000, replace=TRUE),
  column=sample(10, 1000, replace=TRUE)
)

unsplitAsDataFrame(mat)
```

# Index

!, BumpyAtomicMatrix-method  
     (BumpyAtomicMatrix), 2  
 [, BumpyDataFrameMatrix, ANY, ANY, ANY-method  
     (BumpyDataFrameMatrix), 3  
 [, BumpyDataFrameMatrix, ANY-method  
     (BumpyDataFrameMatrix), 3  
 [, BumpyDataFrameMatrix, BumpyMatrix, ANY, ANY-method  
     (BumpyDataFrameMatrix), 3  
 [, BumpyDataFrameMatrix, BumpyMatrix-method  
     (BumpyDataFrameMatrix), 3  
 [, BumpyMatrix, ANY, ANY, ANY-method  
     (BumpyMatrix), 4  
 [, BumpyMatrix, ANY-method (BumpyMatrix),  
     4  
 [, BumpyMatrix, BumpyMatrix, ANY, ANY-method  
     (BumpyMatrix), 4  
 [, BumpyMatrix, BumpyMatrix-method  
     (BumpyMatrix), 4  
 [<-, BumpyDataFrameMatrix, ANY, ANY, BumpyMatrix-method  
     (BumpyDataFrameMatrix), 3  
 [<-, BumpyMatrix, ANY, ANY, BumpyMatrix-method  
     (BumpyMatrix), 4  
 [<-, BumpyMatrix, ANY, ANY, CompressedList-method  
     (BumpyMatrix), 4  
  
 BumpyAtomicMatrix, 2, 9  
 BumpyAtomicMatrix-class  
     (BumpyAtomicMatrix), 2  
 BumpyCharacterMatrix-class  
     (BumpyAtomicMatrix), 2  
 BumpyDataFrameMatrix, 3, 9  
 BumpyDataFrameMatrix-class  
     (BumpyDataFrameMatrix), 3  
 BumpyIntegerMatrix, 5  
 BumpyIntegerMatrix-class  
     (BumpyAtomicMatrix), 2  
 BumpyLogicalMatrix-class  
     (BumpyAtomicMatrix), 2  
 BumpyMatrix, 2, 4, 4, 7  
 BumpyMatrix-class (BumpyMatrix), 4  
 BumpyNumericMatrix-class  
     (BumpyAtomicMatrix), 2  
  
 cbind, BumpyMatrix-method (BumpyMatrix),  
     4  
 chartr, ANY, ANY, BumpyCharacterMatrix-method  
     (BumpyAtomicMatrix), 2  
 commonColNames, BumpyDataFrameMatrix-method  
     (BumpyDataFrameMatrix), 3  
 commonColNames<-, BumpyDataFrameMatrix-method  
     (BumpyDataFrameMatrix), 3  
 CompressedList, 4-7  
 CompressedSplitDataFrameList, 4  
 cor, BumpyAtomicMatrix, BumpyAtomicMatrix-method  
     (BumpyAtomicMatrix), 2  
 cov, BumpyAtomicMatrix, BumpyAtomicMatrix-method  
     (BumpyAtomicMatrix), 2  
  
 DataFrame, 3, 8, 9  
 DFrame, 5, 7  
 dim, BumpyMatrix-method (BumpyMatrix), 4  
 dimnames, BumpyMatrix-method  
     (BumpyMatrix), 4  
 dimnames<-, BumpyMatrix, ANY-method  
     (BumpyMatrix), 4  
 endsWith, BumpyCharacterMatrix-method  
     (BumpyAtomicMatrix), 2  
  
 grep, ANY, BumpyCharacterMatrix-method  
     (BumpyAtomicMatrix), 2  
 grepl, ANY, BumpyCharacterMatrix-method  
     (BumpyAtomicMatrix), 2  
 gsub, ANY, ANY, BumpyCharacterMatrix-method  
     (BumpyAtomicMatrix), 2  
  
 IQR, BumpyAtomicMatrix-method  
     (BumpyAtomicMatrix), 2  
  
 length, BumpyMatrix-method  
     (BumpyMatrix), 4  
 lengths, BumpyMatrix-method  
     (BumpyMatrix), 4  
  
 mad, BumpyAtomicMatrix-method  
     (BumpyAtomicMatrix), 2  
 Math, 2  
 Math, BumpyAtomicMatrix-method  
     (BumpyAtomicMatrix), 2

- Math2, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- mean, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- median, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- nchar, BumpyCharacterMatrix-method  
(BumpyAtomicMatrix), [2](#)
- Ops, [2](#)
- Ops, atomic, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- Ops, BumpyAtomicMatrix, atomic-method  
(BumpyAtomicMatrix), [2](#)
- Ops, BumpyAtomicMatrix, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- Ops, BumpyAtomicMatrix, matrix-method  
(BumpyAtomicMatrix), [2](#)
- Ops, BumpyAtomicMatrix, missing-method  
(BumpyAtomicMatrix), [2](#)
- Ops, matrix, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- pmax, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- pmax.int, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- pmin, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- pmin.int, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- quantile, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- range, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- rbind, BumpyMatrix-method (BumpyMatrix),  
[4](#)
- redim (BumpyMatrix), [4](#)
- redim, CompressedList, BumpyMatrix-method  
(BumpyMatrix), [4](#)
- relist, [6](#)
- sd, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- show, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- show, BumpyDataFrameMatrix-method  
(BumpyDataFrameMatrix), [3](#)
- show, BumpyMatrix-method (BumpyMatrix), [4](#)
- splitAsBumpyMatrix, [7](#), [9](#)
- splitAsList, [7](#)
- startsWith, BumpyCharacterMatrix-method  
(BumpyAtomicMatrix), [2](#)
- sub, ANY, ANY, BumpyCharacterMatrix-method  
(BumpyAtomicMatrix), [2](#)
- substr, BumpyCharacterMatrix-method  
(BumpyAtomicMatrix), [2](#)
- substring, BumpyCharacterMatrix-method  
(BumpyAtomicMatrix), [2](#)
- Summary, [2](#)
- Summary, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- t, BumpyMatrix-method (BumpyMatrix), [4](#)
- tolower, BumpyCharacterMatrix-method  
(BumpyAtomicMatrix), [2](#)
- toupper, BumpyCharacterMatrix-method  
(BumpyAtomicMatrix), [2](#)
- undim (BumpyMatrix), [4](#)
- undim, BumpyMatrix-method (BumpyMatrix),  
[4](#)
- unlist, BumpyMatrix-method  
(BumpyMatrix), [4](#)
- unsplitAsDataFrame, [7](#), [8](#)
- unstrsplit, BumpyCharacterMatrix-method  
(BumpyAtomicMatrix), [2](#)
- var, BumpyAtomicMatrix, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- var, BumpyAtomicMatrix, missing-method  
(BumpyAtomicMatrix), [2](#)
- Vector, [4](#), [6](#), [7](#)
- which.max, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)
- which.min, BumpyAtomicMatrix-method  
(BumpyAtomicMatrix), [2](#)