# Package ‘implicitExpansion’

October 13, 2022

**Version** 0.1.0

**Title** Array Operations for Arrays of Mismatching Sizes

**Description** Support for implicit expansion of arrays in operations involving arrays of mismatching sizes. This pattern is known as ``broadcasting'' in 'Python' and ``implicit expansion'' in 'Matlab' and is explained for example in the article ``Array programming with NumPy'' by C. R. Harris et al. (2020) <doi:10.1038/s41586-020-2649-2>.

**License** MIT + file LICENSE

**URL** `https://github.com/ManuelHentschel/implicitExpansion`

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Manuel Hentschel [aut, cre]

**Maintainer** Manuel Hentschel <`Manuel.Hentschel@unige.ch`>

**Repository** CRAN

**Date/Publication** 2022-10-02 23:20:02 UTC

## R topics documented:

---

## implicitExpansion-package *Implicit Expansion*

---

### Description

This package implements a feature known as "Broadcasting" in Python (see e.g. here) and as "Implicit Expansion" in Matlab (see e.g. here). In operations involving multiple arguments of type array (or vector, matrix, list) with mismatching dimensions, any argument is repeated along its dimensions of size (exactly) 1, as often as necessary to match the other argument(s).

### Details

Below are some examples that illustrate possible operations using implicit expansion. For detailed explanations of the behavior, see the corresponding docs for Python and Matlab.

Dimensions of size 0 are ok, as long as all other arrays are also of size 0 or 1 in that dimension.

All arguments to an operation with implicit expansion are coerced to arrays first.

Currently, all arguments to an operation with implicit expansion are expanded to the full size of the output first, resulting in bad performance for very large arrays.

The package `rray` ( on GitHub, documentation, currently archived on CRAN ) provides similar functionality as part of complete a remodeling of the array object.

### See Also

`mmapply`, `BinaryOperators`, `expandArray`, `expandedDim`, `ArrayCreation`, `RowAndColumnVectors`

### Examples

```
x <- c(1,2,3)
y <- t(c(4,5))
x %m+% y
mmapply(sum, x, x, t(x))

m <- matrix(3*(1:12)^2, 3, 4)
cm <- t(colMeans(m))
m %m-% cm

summaries <- list(Max = max, Min = min, avg = mean)
data <- list(a = 1:5, b = 2:3, c = 20:12)
formatStrings <- array(c('%.1f', '%.3f'), c(1,1,2))
mmapply(function(f, d, s) sprintf(s, f(d)), summaries, t(data), formatStrings)
```

## `as.mRowVector` *Row and Column Vectors*

### Description

These functions coerce an object to a 2-dimensional `array` in the shape of a row or column vector, i.e. an array with dimensions `c(1,d)` or `c(d,1)` respectively, where `d=length(x)`.

### Usage

```
as.mRowVector(x, USE.NAMES = TRUE)

as.mColVector(x, USE.NAMES = TRUE)
```

### Arguments

| | |
|---|---|
| `x` | An object that can be passed to `array()`. |
| `USE.NAMES` | If `TRUE` and `x` has only one non-singular dimension, its names are preserved. |

### Value

A row or column vector with the data from `x`.

### Examples

```
x <- c(a=1, b=2, c=3)
as.mColVector(x)

y <- matrix(1:12, 3, 4)
as.mRowVector(y)
```

## `expandArray` *Expand an Array to a Given Dimension*

### Description

Expand an array to a given dimension by repeating its entries in the directions where the array has dimension one.

### Usage

```
expandArray(x, dy)
```

## Arguments

| | |
|---|---|
| `x` | An object that is coerced to array. |
| `dy` | The dimensions it is to be expanded to. |

## Details

Throws an error if the array and dimensions are not compatible.

## Value

An array that consists of the entries in `x`, with dimension `dy`.

## See Also

`expandedDim`, `mmapply`

## Examples

```
x <- 1:3
expandArray(x, c(3,4))
```

---

`expandedDim` *Implied Dimension of a set of Arrays*

---

## Description

Get the dimension implied by a set of arrays as used in implicit expansion.

## Usage

```
expandedDim(..., arrays = list(), allowRecycling = FALSE)
```

## Arguments

| | |
|---|---|
| `...` | Objects that are coerced to arrays. |
| `arrays` | A list of objects that are coerced to arrays. |
| `allowRecycling` | Whether to allow recycling of elements in each dimension. |

## Details

Both the arrays in `...` and `arrays` are considered by concatenating them with `c(list(...), arrays)`. Throws an error if the arrays are not compatible.

## Value

A numberical vector containing the expanded dimension implied by the arrays.

## See Also

`expandArray`

## Examples

```
x <- 1:3
y <- t(4:5)
z <- array(0, c(1,1,6))
expandedDim(x, y, z)
```

## mmapply — *Apply a Function to Multiple Arrays with Implicit Expansion*

### Description

Similar function to `mapply` with support for implicit expansion.

### Usage

```
mmapply(
  FUN,
  ...,
  MoreArgs = list(),
  SIMPLIFY = TRUE,
  USE.NAMES = TRUE,
  ALLOW.RECYCLING = FALSE
)
```

### Arguments

| | |
|---|---|
| `FUN` | Function to apply to each combination of arguments. Found via `match.fun`. |
| `...` | Objects that are coerced to arrays, expanded using implicit expansion, and then vectorized over. |
| `MoreArgs` | Pass arguments in a list instead of `...` |
| `SIMPLIFY` | If `TRUE`, the resulting list array is simplified to an atomic array if possible. |
| `USE.NAMES` | If `TRUE`, the dimensions are named using the names of input arguments of matching size. |
| `ALLOW.RECYCLING` | Whether to allow recycling of elements in each dimension. |

### Details

Most arguments are handled in a similar fashion to `mapply` with some key differences:

- Entries of `MoreArgs` are treated the same as the ones in `...`, i.e. `mmapply(...)` is the same as `mmapply(MoreArgs = list(...))`. Additional arguments to FUN can be passed here or in `...`, either as they are (if they are atomic), or as a `list()` of length one.
- `SIMPLIFY` only simplifies a list array to an atomic array, nothing else.
- `USE.NAMES` uses names from all arguments, but never uses an argument itself as names.

If `ALLOW.RECYCLING` is set to `TRUE`, all arrays of any size are compatible.

### Value

An array containing the result of `FUN` for each combination of entries from `...` after implicit expansion.

### See Also

`expandArray`, `expandedDim`

### Examples

```
summaries <- list(Max = max, Min = min, avg = mean)
data <- list(a = 1:5, b = 2:3, c = 20:12)
formatStrings <- array(c('%.1f', '%.3f'), c(1,1,2))
mmapply(function(f, d, s) sprintf(s, f(d)), summaries, t(data), formatStrings)
```

---

## `mZeros` *Array Creation*

---

### Description

Convenience functions that create an array full of identical entries.

### Usage

```
mZeros(...)

mOnes(...)

mTRUE(...)

mFALSE(...)

mNULL(...)
```

### Arguments

| | |
|---|---|
| `...` | Numbers or numeric vectors, passed to `c()` and then used as `dim` argument in a call to `array()`. |

### Value

The result of `array(XXX, c(...))`, where XXX is `0`, `1`, `TRUE`, `FALSE`, or `list()`, respectively.

### Examples

```
mZeros(2, 3)

mOnes(c(1, 2, 3))

mTRUE(c(1, 3), 2)

mFALSE(5)
```

---

## `%m+%` *Binary Operators with Implicit Expansion*

---

### Description

Modified versions of binary operators that support implicit expansion. Arguments are passed to `mmapply()` with the corresponding element-wise binary operator. For instance, `x %m+% y` is equivalent to `mmapply('+', x, y)`.

### Usage

```
x %m+% y

x %m-% y

x %m*% y

x %m/% y

x %m^% y

x %m==% y

x %m!=% y

x %m>% y

x %m<% y
```

```
x %m>=% y

x %m<=% y

x %m|% y

x %m&% y
```

## Arguments

| | |
|---|---|
| `x, y` | Arrays or objects that can be coerced to arrays using `as.array()`. |

## Value

The result of `mmapply('XXX', x, y)`, with `XXX` replaced by the corresponding element-wise binary operator.

## See Also

`mmapply`

## Examples

```
x <- c(1,2,3)
y <- t(c(4,5))
x %m+% y

m <- matrix(3*(1:12)^2, 3, 4)
cm <- t(colMeans(m))
m %m-% cm

(1:3) %m>=% t(3:1)
```

# Index