

Package ‘lagged’

July 22, 2025

Type Package

Title Classes and Methods for Lagged Objects

Version 0.3.2

Date 2022-08-07

Maintainer Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

Description Provides classes and methods for objects, whose indexing naturally starts from zero. Subsetting, indexing and mathematical operations are defined naturally between lagged objects and lagged and base R objects. Recycling is not used, except for singletons. The single bracket operator doesn't drop dimensions by default.

URL <https://github.com/GeoBosh/lagged> (devel),
<https://geobosh.github.io/lagged/> (doc)

BugReports <https://github.com/GeoBosh/lagged/issues>

Imports methods

Suggests testthat

License GPL (>= 2)

LazyLoad yes

Collate lagged.R utils.R pc20slMatrix.r acfutils.R

RoxygenNote 6.1.1

NeedsCompilation no

Author Georgi N. Boshnakov [aut, cre]

Repository CRAN

Date/Publication 2022-08-07 15:20:21 UTC

Contents

acf2Lagged	2
dataWithLagNames	3
FlexibleLagged-class	4

Lagged	6
Lagged-class	9
Lagged1d-class	10
Lagged2d-class	11
Lagged3d-class	12
maxLag	13
maxLag<-	15
nSeasons	16
sl2acfbase	17
slMatrix	18
slMatrix-class	20
[-methods	21
[<-methods	22
[[-methods	23
[[<-methods	24
Index	25

acf2Lagged	<i>Convert "acf" objects to "Lagged" objects</i>
------------	--

Description

Convert "acf" objects to "Lagged" objects.

Usage

acf2Lagged(x)

Arguments

x an object from "S3" class "acf", typically obtained from acf() and related functions.

Details

acf2Lagged() converts objects produced by acf() and friends to suitable "Lagged" objects. Partial autocorrelations obtained from acf() do not contain value for lag zero. acf2Lagged() puts the number 1 at lag zero in the univariate case and a matrix of NA's in the multivariate case.

Value

an object from class "Lagged1d" (univariate case) or "Lagged3d" (multivariate case)

Author(s)

Georgi N. Boshnakov

Examples

```
## using examples from help(acf)
lh_acf <- acf2Lagged(acf(lh))

lh_acf[0:5]
acf(lh, plot = FALSE)$acf[1 + 0:5] # same

acf(ts.union(mdeaths, fdeaths))$acf[15,,]

deaths_mts <- ts.union(mdeaths, fdeaths)
deaths_acf <- acf2Lagged(acf(deaths_mts))
base_acf <- acf(deaths_mts)

## rho_14
deaths_acf[14]
base_acf$acf[1 + 14, , ] # same
## this is different and maybe surprising to some:
base_acf[14]
## (see also examples in \link{Lagged})
```

dataWithLagNames	<i>Lagged data with named lag dimension</i>
------------------	---

Description

Get the data from a Lagged object and ensure that the lag dimension is named

Usage

```
dataWithLagNames(object, prefix = "Lag_")
```

Arguments

object	an object inheriting from "Lagged".
prefix	a character string.

Details

dataWithLagNames() extracts the data part from a lagged object and gives names to the lag dimension, if it is not already named.

This function is mainly used for programming, particularly in show() methods for lagged objects..

Value

The data part with names as described in Details.

Author(s)

Georgi N. Boshnakov

Examples

```
x <- Lagged(drop(acf(ldeaths, plot = FALSE)$acf))
## there are no names for the lags:
names(x) # NULL
## but the print method inserts default "Lag_N" names
x
## This sets the names to their defaults:
x1 <- dataWithLagNames(x)
names(x1)
## ... and this sets non-default prefix:
x2 <- dataWithLagNames(x, "L")
names(x2)
x2
```

FlexibleLagged-class *Class FlexibleLagged*

Description

Class FlexibleLagged.

Objects from the Class

Objects can be created by calls of the form `new("FlexibleLagged", data, ...)`, see also convenience function [Lagged](#),

"FlexibleLagged" is used mainly in programming as a superclass of classes which need to inherit from all "Lagged" classes. It can represent objects from any subclass of "Lagged". Methods are defined, such that the internal representation is transparent.

Slots

data: Object of class "Lagged" ~~

Extends

Class "[Lagged](#)", directly.

Methods

```
[ signature(x = "FlexibleLagged", i = "ANY"): ...
[ signature(x = "FlexibleLagged", i = "missing"): ...
[<- signature(x = "FlexibleLagged", i = "missing"): ...
[<- signature(x = "FlexibleLagged", i = "numeric"): ...
```

Author(s)

Georgi N. Boshnakov

See Also

[Lagged](#), [Lagged1d](#), [Lagged2d](#), [Lagged3d](#)

Examples

```
## Lagged1d
v <- 1:12
v_lagged <- Lagged(v)
v_lagged
identical(v_lagged, new("Lagged1d", data = v)) # TRUE
v_lagged[0:2] # v[1:3]
v_lagged[[0]] # 1

## Lagged2d
m <- matrix(1:12, nrow = 4)
m_lagged <- Lagged(m)
m_lagged
identical(m_lagged, new("Lagged2d", data = m)) # TRUE
m_lagged[0] # matrix with 1 column
m_lagged[[0]] # numeric

## Lagged3d
a <- array(1:24, dim = c(2, 3, 4))
a_lagged <- Lagged(a)
identical(a_lagged, new("Lagged3d", data = a)) # TRUE

dim(a_lagged[0]) # c(2,3,1)
a_lagged[0]
a[, , 1, drop = FALSE]

dim(a_lagged[[0]]) # c(2,3)
a_lagged[[0]]
a[, , 1, drop = TRUE]

## as above "FlexibleLagged"
## 1d
v_flex <- new("FlexibleLagged", data = v)
identical(v_flex@data, v_lagged) # TRUE
v_flex[0] # = v_lagged[0]
v_flex[[0]] # = v_lagged[[0]]

## 2d
m_flex <- new("FlexibleLagged", data = m)
identical(m_flex@data, m_lagged) # TRUE
m_flex[0] # = m_lagged[0]
m_flex[[0]] # = m_lagged[[0]]

## 3d
a_flex <- new("FlexibleLagged", data = a)
identical(a_flex@data, a_lagged) # TRUE
a_flex[0] # = a_lagged[0]
a_flex[[0]] # = a_lagged[[0]]
```

Lagged

Create Lagged objects

Description

Create objects inheriting from Lagged.

Usage

```
Lagged(data, ...)
```

Arguments

data	suitable data, see Details.
...	further arguments passed on to new().

Details

Lagged creates an object inheriting from "Lagged". The exact class depends on argument data. This is the easiest way to create lagged objects.

If data is a vector, matrix or 3D array, the result is "Lagged1d", "Lagged2d" and "Lagged3d", respectively. If data inherits from "Lagged", the result is "FlexibleLagged".

Value

a suitable "Lagged" object, as described in Details

Note

I am considering making Lagged generic.

Author(s)

Georgi N. Boshnakov

See Also

the specific classes [Lagged1d](#), [Lagged2d](#), [Lagged3d](#)

Examples

```
## a discrete distribution with outcomes 0,1,2,3,4,5,6:
v <- dbinom(0:6, size = 6, p = 0.5)
bin6 <- Lagged(v)

## names are used, if present:
names(v) <- paste0("P(x=", 0:6, ")")
bin6a <- Lagged(v)
```

```

bin6a
## subsetting drops the laggedness:
bin6a[1:3]
bin6a[]
bin6a[0:2]
## to resize (shrink or extend) the object use 'maxLag<-':
s8 <- s2 <- bin6a
maxLag(s2) <- 2
s2
## extending inserts NA's:
maxLag(s8) <- 8
s8
## use assignment to extend with specific values:
s8a <- bin6a
s8a[7:8] <- c("P(x=7)" = 0, "P(x=8)" = 0)
s8a

## adapted from examples for acf()
## acf of univariate ts
acv1 <- acf(ldeaths, plot = FALSE)
class(acv1) # "acf"
a1 <- drop(acv1$acf)
class(a1) # numeric

la1 <- Lagged(a1) # 1d lagged object from 'numeric':
Lagged(acv1)      # 1d lagged object from 'acf':

## acf of multivariate ts
acv2 <- acf(ts.union(mdeaths, fdeaths), plot = FALSE)
class(acv2) # "acf"
a2 <- acv2$acf
class(a2) # 3d array
dim(a2)

la2a <- acf2Lagged(acv2) # explicitly convert 'acf' to lagged object
Lagged(acv2)             # equivalently, just use Lagged()
identical(la2a, Lagged(acv2)) # TRUE

la2a[0]    # R_0, indexing lagged object
a2[1, , ]  # same, indexing array data from 'cf' object
acv2[0]     # same, indexing 'acf' object

la2a[1]     # R_1
a2[2, , ]   # same
acv2[1/12]  # transposed, see end of this example section as to why use 1/12

la2a[2]     # R_1
a2[3, , ]   # same
acv2[2/12]  # transposed, see end of this example section as to why use 1/12

## multiple lags
la2a[0:1]    # native indexing with lagged object
a2[1:2, , ]  # different ordering

```

```

acv2$acf[1:2, ,] # also different ordering

## '[' doesn't drop a dimension even when it is of length 1:
la2a[0]
la2a[1]

## to get a singleton element, use '[[':
la2a[[0]]
la2a[[1]]

## arithmetic and math operators
-la1
+la1

2*la1
la1^2
la1 + la1^2

abs(la1)
sinpi(la1)
sqrt(abs(la1))

## Summary group
max(la1)
min(la1)
range(la1)
prod(la1)
sum(la1)
any(la1 < 0)
all(la1 >= 0)

## Math2 group
round(la1)
round(la1, 2)
signif(la1)
signif(la1, 4)

## The remaining examples below are only relevant
## for users comparing to indexing of 'acf'
##
## indexing in base R acf() is somewhat mysterious, so an example with
## DIY computation of lag_1 autocovariance matrix
n <- length(mdeaths)
tmpcov <- sum((mdeaths - mean(mdeaths)) * (fdeaths - mean(fdeaths)) ) / n
msd <- sqrt(sum((mdeaths - mean(mdeaths))^2)/n)
fsd <- sqrt(sum((fdeaths - mean(fdeaths))^2)/n)
tmpcov1 <- sum((mdeaths - mean(mdeaths))[2:n] *
              (fdeaths - mean(fdeaths))[1:(n-1)] ) / n
tmpcov1 / (msd * fsd)

la2a[[1]][1,2] - tmpcov1 / (msd * fsd) # only numerically different
## the raw acf in the 'acf' object is not surprising:
la2a[[1]][1,2] == acv2$acf[2, 1, 2] # TRUE

```

```
## ... but this probably is:
acv2[1]
## the ts here is monthly but has unit of lag 'year'
## so acv2[1] asks for lag 1 year = 12 months, thus
acv2[1/12]
all( acv2$acf[13, , ] == drop(acv2[1]$acf) ) # TRUE
all( acv2$acf[2, , ] == drop(acv2[1/12]$acf) ) # TRUE
all( la2a[[1]] == drop(acv2[1/12]$acf) ) # TRUE
```

Lagged-class

Class *Lagged*

Description

Class Lagged.

Objects from the Class

This class serves as a base class for objects with natural indexing starting from zero. It is a virtual class, no objects can be created from it.

Arithmetic and other operations are defined. They return objects strictly from the core "Lagged" classes, even if the arguments are from classes inheriting from the core "Lagged" classes. Of course, for such classes specialised methods can be defined to keep the class when appropriate. For example, the sum of two autocovariance functions is an autocovariance function, but their difference may not be a valid one.

In arithmetic operations between "Lagged" objects the arguments are made of equal length by filling in NA's. When one of the operands is not "Lagged", the recycling rule is applied only if that argument is a singleton.

Slots

data: Object of class "ANY". Subclasses of "Lagged" may restrict the class of this slot.

Methods

[signature(x = "Lagged", i = "missing", j = "ANY", drop = "ANY"): In this case (i.e., i is missing) [], returns the underlying data. This is equivalent to using x[1:maxLag(x)].

maxLag signature(object = "Lagged"): Gives the maximal lag in the object.

[[signature(x = "Lagged", i = "numeric"): ...

[[<- signature(x = "Lagged", i = "numeric"): ...

[<- signature(x = "Lagged", i = "missing"): ...

coerce signature(from = "Lagged", to = "array"): ...

coerce signature(from = "Lagged", to = "matrix"): ...

```

coerce signature(from = "Lagged", to = "vector"): ...
Math signature(x = "Lagged"): ...
Math2 signature(x = "Lagged"): ...
maxLag<- signature(object = "Lagged"): ...
Ops signature(e1 = "FlexibleLagged", e2 = "Lagged"): ...
Ops signature(e1 = "Lagged", e2 = "FlexibleLagged"): ...
Ops signature(e1 = "Lagged", e2 = "Lagged"): ...
Ops signature(e1 = "Lagged", e2 = "missing"): ...
Ops signature(e1 = "Lagged", e2 = "vector"): ...
Ops signature(e1 = "vector", e2 = "Lagged"): ...
Summary signature(x = "Lagged"): ...

```

Author(s)

Georgi N. Boshnakov

See Also

function [Lagged](#) which creates objects from suitable subclasses of "Lagged", and also [Lagged1d](#), [Lagged2d](#), [Lagged3d](#)

Examples

```

Lagged(1:12)                # => Lagged1d
Lagged(matrix(1:12, ncol = 3)) # => Lagged2d
Lagged(array(1:24, dim = 2:4)) # => Lagged3d

## equivalently:
new("Lagged1d", data = 1:12)    # => Lagged1d
new("Lagged2d", data = matrix(1:12, ncol = 3)) # => Lagged2d
new("Lagged3d", data = array(1:24, dim = 2:4)) # => Lagged3d

```

Lagged1d-class

Class Lagged1d

Description

Class Lagged1d.

Objects from the Class

Objects can be created by calls of the form `Lagged(v)` or `new("Lagged1d", data = v)`, where `v` is a vector. `new("Lagged1d", ...)` also works.

Slots

data: Object of class "vector".

Extends

Class "[Lagged](#)", directly.

Methods

```
[<- signature(x = "Lagged1d", i = "numeric"): ...
[ signature(x = "Lagged1d", i = "numeric", j = "ANY", drop = "ANY"): ...
show signature(object = "Lagged1d"): ...
whichLagged signature(x = "Lagged1d", y = "missing"): ...
```

Author(s)

Georgi N. Boshnakov

See Also

[Lagged](#), [Lagged2d](#), [Lagged3d](#)

Examples

```
v <- cos(2*pi*(0:10)/10)
new("Lagged1d", data = v) ## ok, but Lagged() is more convenient
x <- Lagged(v)
class(x) # Lagged1d
x
x[0]
x[0:3]
```

Lagged2d-class

Class Lagged2d

Description

Class Lagged2d.

Objects from the Class

Objects can be created by calls of the form `Lagged(m)` or `new("Lagged2d", data = m)`, where `m` is a matrix. `new("Lagged2d", ...)` also works.

Slots

data: Object of class "matrix" ~~

Extends

Class "[Lagged](#)", directly.

Methods

```
[ signature(x = "Lagged2d", i = "numeric", j = "missing", drop = "logical"): ...
[ signature(x = "Lagged2d", i = "numeric", j = "missing", drop = "missing"): ...
[<- signature(x = "Lagged2d", i = "numeric"): ...
show signature(object = "Lagged2d"): ...
whichLagged signature(x = "Lagged2d", y = "missing"): ...
```

Author(s)

Georgi N. Boshnakov

See Also

[Lagged](#), [Lagged1d](#), [Lagged3d](#)

Examples

```
powers <- Lagged(outer(1:6, 0:6, `^`))
powers[[0]]
powers[[1]]
powers[[2]]
```

Lagged3d-class	<i>Class Lagged3d</i>
----------------	-----------------------

Description

Class Lagged3d.

Objects from the Class

Objects can be created by calls of the form `Lagged(a)` or `new("Lagged3d", data = a)`, where `a` is a 3-d array. `new("Lagged3d", ...)` also works.

Slots

`data`: Object of class "array" ~~

Extends

Class "[Lagged](#)", directly.

Methods

```
[ signature(x = "Lagged3d", i = "numeric", j = "missing", drop = "logical"): ...
[ signature(x = "Lagged3d", i = "numeric", j = "missing", drop = "missing"): ...
[<- signature(x = "Lagged3d", i = "numeric"): ...
show signature(object = "Lagged3d"): ...
whichLagged signature(x = "Lagged3d", y = "missing"): ...
```

Author(s)

Georgi N. Boshnakov

See Also

[Lagged](#), [Lagged1d](#), [Lagged2d](#)

Examples

```
## see examples for class "Lagged"
```

maxLag	<i>Give the maximal lag in an object</i>
--------	--

Description

Give the maximal lag in an object, such as autocorrelations.

Usage

```
maxLag(object, ...)
```

Arguments

object	an object, for which the function makes sense.
...	not used?

Details

maxLag is a generic function to get the maximal lag for which information is available in lagged objects.

Value

a non-negative integer

Methods

signature(object = "Lagged") This method applies to all classes inheriting from "Lagged".

signature(object = "array")

signature(object = "matrix")

signature(object = "vector")

signature(object = "ANY")

signature(object = "slMatrix")

Author(s)

Georgi N. Boshnakov

See Also

["maxLag<-"](#)

Examples

```
## 1d
v <- Lagged(2^(0:6))
v
maxLag(v)
v[c(2,4,6)]
v[8] # NA
## reduce the number of lags in place
maxLag(v) <- 4
v
## extend the object (with NA's)
maxLag(v) <- 6
v
## extend using "["
v[5:8] <- 2^(5:8)
v

## 2d
m <- Lagged(matrix(1:12, nrow = 4))
m
maxLag(m)
maxLag(m) <- 1
m
## maxLag(m) <- 4 # extending this way doesn't work currently
```

maxLag<-*Change the maximal lag in a lagged object*

Description

Change the maximal lag in a lagged object.

Usage

```
maxLag(object, ...) <- value
```

Arguments

object	an object for which this makes sense.
...	currently not used.
value	the new value of the maximal lag, a non-negative integer number.

Details

The replacement version of maxLag() changes the maximal lag in an object to value. It is a generic function with no default method.

For the core Lagged classes this is done by simply extending or shrinking the data part to the requested value. Subclasses of "Lagged" and other classes, in general, may need more elaborate changes. If so, they should define their own methods.

When value is larger than the current maxLag(object), the entries for the new lags are filled with NA's.

Value

the object whose maximal lag is modified as described in Details.

Methods

```
signature(object = "Lagged")  
signature(object = "FlexibleLagged")
```

Author(s)

Georgi N. Boshnakov

See Also

[maxLag](#)

Examples

```

la1 <- Lagged(drop(acf(ldeaths)$acf))
la3 <- la1
la3
## shrink la3
maxLag(la3) # 18
maxLag(la3) <- 5
la3
maxLag(la3) # 5

## extend la3, new entries are filled with NA's
maxLag(la3) <- 10
la3

## alternatively, use "[<-" which accepts any replacement values
la3[11:13] <- 0
la3

```

nSeasons

Get the number of seasons from an object

Description

Get the number of seasons from an object.

Usage

```

nSeasons(object)
nSeasons(object, ...) <- value

```

Arguments

object	an object for which the notion of number of seasons makes sense.
value	a positive integer number.
...	further arguments for methods.

Details

These are generic functions.

Methods for nSeasons are straightforward when the property makes sense for objects from a class. In contrast, methods for the replacement version, ``nSeasons<=``, should be defined carefully and may not even be feasible.

Value

an integer number

Methods

No methods for ``nSeasons<-`` are defined in package **lagged**. The methods defined for `nSeasons` are given below.

```
signature(object = "slMatrix")
```

Author(s)

Georgi N. Boshnakov

Examples

```
m <- slMatrix(matrix(1:12, nrow = 4))
m
nSeasons(m)
```

sl2acfbase	<i>Convert between vector and season-lag representations</i>
------------	--

Description

Convert between vector and season-lag representations of autocovariances of multivariate and periodically correlated time series.

Usage

```
sl2acfbase(mat, maxlag, fullblocks = FALSE)

acfbase2sl(acf)

sl2vecacf(mat, maxlag, fullblocks = FALSE)
```

Arguments

<code>acf</code>	an <code>acf</code> as returned by base R <code>acf</code> .
<code>mat</code>	a matrix containing autocovariances in season-lag arrangement.
<code>maxlag</code>	maximal lag, a positive integer.
<code>fullblocks</code>	if TRUE, keep full blocks only.

Details

These functions rearrange autocovariances and autocorrelations between the native season-lag arrangement in package “pcts” and the vector representations of the corresponding multivariate models (vector of seasons representation of periodic models). Variable s is taken be season s and vice versa in the opposite direction.

“acfbase” in the names of the functions refers to the representation returned by base function `acf`.

acfbase2sl rearranges a multivariate acf in season-lag form.

sl2acfbase rearranges a season-lag form into the multivariate form used by base function acf.

sl2vecacf is similar to sl2acfbase but the result is such that the lag is in the third dimension and $r[, , k]$ is $Cov(X_t, X_{t-k})$ (not its transpose). See also the examples below and in [acf2Lagged](#).

Value

for acfbase2sl, a matrix.

for sl2acfbase and sl2vecacf, an array.

Author(s)

Georgi N. Boshnakov

Examples

```
## use a character matrix to illustrate the positions of the elements
matsl <- rbind(paste0("Ra", 0:3), paste0("Rb", 0:3))
matsl
## convert to what I consider "standard" vec format R(k)=EX_tX_{t-k}'
sl2vecacf(matsl)
## convert to the format from acf() (R(k) is the transposed from mine).
sl2acfbase(matsl)
identical(sl2vecacf(matsl), aperm(sl2acfbase(matsl), c(3, 2, 1))) # TRUE

## by default the conversion is lossles;
## so this contains all values from the original and some NA's:
sl2acfbase(matsl)
## the original, matsl, can be restored:
acfbase2sl(sl2acfbase(matsl))
identical(acfbase2sl(sl2acfbase(matsl)), matsl) # TRUE

## this drops some values (if necessary) to keep complete block only
sl2acfbase(matsl, fullblocks = TRUE)
```

slMatrix

Function to create objects from class slMatrix

Description

Provides a flexible way to create objects from class slMatrix. The entries may be specified in several ways.

Usage

```
slMatrix(init = NA, period, maxlag, seasonnames = seq(length = period),
         lagnames = as.character(0:maxlag), periodunit = "season",
         lagunit = "lag", f = NA, type = "sl")
```

Arguments

<code>init</code>	values for the the autocovariances, see also argument <code>f</code> .
<code>period</code>	the number of seasons in an epoch
<code>maxlag</code>	maximum lag to be stored
<code>seasonnames</code>	names of seasons (?)
<code>lagnames</code>	names of lags
<code>periodunit</code>	name of the period unit
<code>lagunit</code>	name of the unit for lags
<code>f</code>	function to evaluate or matrix to get the values of the autocovariances.
<code>type</code>	format or the arguments of <code>f</code> , see details.

Details

The internal representation of `slMatrix` is a matrix slot, `m`, of size `period` x `(maxlag+1)`. It is created by a call to `matrix()` with `init` supplying the values (may be NAs). If `init` is a matrix values for `period` and `maxlag` are deduced (if not supplied) from its size.

Change on 21/06/2006: Now, if the length of `init` is smaller than that of `m`, the remaining values are filled with NA's (in the past the normal recycling rules of `matrix()` applied). The previous behaviour used to hide puzzling and difficult to track errors. I cannot be sure but this change should not affect old code.

If `f` is given it is used to populate the slot `m` by a call to `fill.slMatrix`. Normally in this case `init=NA` but this is not required.

Currently `fill.slMatrix` has methods for `f` of class "matrix" and "function". The arguments (or the indices) can be controlled by the argument `type`.

`type="sl"` - standard season-lag pair

`type="tt"` - time-time pair

`type="tl"` - standard season-lag pair

Value

An object of class `slMatrix`

Note

To do: additional work is needed on the case when the dimensions of `init` and the result are not the same (see the details section)

Author(s)

Georgi N. Boshnakov

See Also

[slMatrix-class](#), [fill.slMatrix](#)

slMatrix-class

Class "slMatrix"

Description

slMatrix is a matrix-like object for storing values of periodic autocovariance functions, i.e. of functions of two arguments which are periodic in the first argument, $r[t,k]=r[t+d,k]$. The first argument has the meaning of "time" or "season" (when taken modulo the period), the second is "lag". This class provides various access and assignment methods for such objects. slMatrix was created as the storage for values of periodic autocovariance functions and is used for other related quantities.

Objects from the Class

Objects can be created by calls of the form `new("slMatrix", m)`, where `m` is a matrix with `m[i,k]` giving the values for season `i` and lag `(k-1)`, $k = 1, 2, \dots$. The number of rows in `m` is taken to be the number of seasons. The function `slMatrix` provides several ways to specify the data for the slMatrix object.

Slots

`m`: Object of class "matrix".

Methods

`[<- signature(x = "slMatrix", i = "ANY", j = "ANY", value = "ANY"): ...`

`[ signature(x = "slMatrix", i = "ANY", j = "ANY", drop = "ANY"):`

The indexing method is quite flexible and allows to extract parts of slMatrix objects in a variety of ways. It returns an ordinary matrix or, if `drop = TRUE`, vector.

The syntax for indexing is similar to that for ordinary matrices with some features specific to the periodic nature of the first index. The named parameters are `i`, `j`, and `type`. Both `i` and `j` can be vectors. The interpretation of `i` and `j` depends on `type`.

`x[i, j]` (or `x[i, j, type="sl"]`) refers to the value for season `i` and lag `j`. This is referred to as standard season-lag pair, meaning that the elements of `i` must be in the range $1, \dots, d$, where `d` is the number of seasons and the lags must be non-negative. Negative indices have the usual effect of removing the corresponding elements. A zero element for lag is admissible.

`x[i, j, type="tl"]` is similar to "sl" but `i` is allowed to take any (integer) values. These are reduced modulo the number of seasons to the $1, \dots, d$, range.

`x[i, j, type="tl+-"]` This allows also the lags to be negative.

`x[i, j, type="co"]` ("co" stands for "coefficient") This assumes that the values for negative lags and lags larger than `maxlag` are 0. If assignment is attempted for such lags, a message is issued and the assignment is ignored.

`x[i, j, type="tt"]` both arguments have the meaning of time. If `i` and `j` are scalars the pair `i, j` is converted to standard `s, l` pair and the value assigned to the relevant element. If `i` and/or `j` are vectors, they are crossed and the procedure is done for each pair.

If several values need to be assigned to the same s, l pair a warning is issued if these values are not all equal.

Obviously, wherever negative arguments are allowed, elements to omit cannot be specified with negative indices.

see [-methods.

maxLag signature($x = "slMatrix"$): maximum lag available for storage.

Note

The current implementation of the indexing is inefficient, I simply added features and patches as the need arose. Maybe some day I will replace it with C code.

Author(s)

Georgi N. Boshnakov

See Also

[slMatrix](#)

Examples

```
m1 <- rbind(c(1, 0.81, 0), c(1, 0.4972376, 0.4972376))
x <- slMatrix(m1)
x[1, 0]
x[1:2, 0:1]
x[1:3, 1:3, type = "tt"]
```

[-methods

Methods for subsetting defined in package 'lagged'

Description

Methods for subsetting defined in package 'lagged'.

Methods

Subscripting "Lagged" objects always drops the Lagged-ness.

When i is missing, $x[]$, returns the underlying data. This is equivalent to using $x[0:\text{maxLag}(x)]$.

Subscripting (with one index) is defined naturally. It returns the suitably subscripted data slot (for "FlexibleLagged" it is the data slot of the data slot). For indices larger than the maximal lag the values are NA.

Currently negative indices work similarly to the standard R indexing in that negative indices are used to drop elements. However, for $k > 0$, using $-k$ as an index drops the element for lag $k - 2$, not k (since the subsetting is done by something like $x@data[i+1]$). This is implementation detail, so it may be changed and should not be relied upon.

The following methods for "[" are currently defined in package "lagged":

```

signature(x = "FlexibleLagged", i = "missing", j = "ANY", drop = "ANY")
signature(x = "FlexibleLagged", i = "numeric", j = "missing", drop = "logical")
signature(x = "FlexibleLagged", i = "numeric", j = "missing", drop = "missing")
signature(x = "Lagged", i = "missing", j = "ANY", drop = "ANY")
signature(x = "Lagged1d", i = "numeric", j = "ANY", drop = "ANY")
signature(x = "Lagged2d", i = "numeric", j = "missing", drop = "logical")
signature(x = "Lagged2d", i = "numeric", j = "missing", drop = "missing")
signature(x = "Lagged3d", i = "numeric", j = "missing", drop = "logical")
signature(x = "Lagged3d", i = "numeric", j = "missing", drop = "missing")
signature(x = "slMatrix", i = "ANY", j = "ANY", drop = "ANY")
signature(x = "Lagged2d", i = "ANY", j = "ANY", drop = "character")
signature(x = "Lagged2d", i = "missing", j = "numeric", drop = "missing")
signature(x = "Lagged2d", i = "numeric", j = "numeric", drop = "missing")
signature(x = "FlexibleLagged", i = "missing", j = "missing", drop = "ANY")
signature(x = "ANY", i = "ANY", j = "ANY", drop = "ANY")
signature(x = "nonStructure", i = "ANY", j = "ANY", drop = "ANY")
signature(x = "Lagged2d", i = "character", j = "missing", drop = "logical")
signature(x = "Lagged2d", i = "character", j = "missing", drop = "missing")
signature(x = "Lagged2d", i = "character", j = "character", drop = "missing")
signature(x = "Lagged2d", i = "missing", j = "character", drop = "missing")
signature(x = "Lagged2d", i = "numeric", j = "character", drop = "missing")
signature(x = "Lagged2d", i = "character", j = "numeric", drop = "missing")

```

[<-methods

*Methods for subset assignment***Description**

Methods for subset assignment.

Methods

```

signature(x = "FlexibleLagged", i = "missing")
signature(x = "FlexibleLagged", i = "numeric")
signature(x = "Lagged1d", i = "numeric")
signature(x = "Lagged", i = "missing")
signature(x = "Lagged2d", i = "numeric")
signature(x = "Lagged3d", i = "numeric")
signature(x = "slMatrix", i = "ANY")

```

[[-methods

*Methods for '[' in package 'lagged'***Description**

Methods for '[' in package 'lagged'.

Methods

Indexing with "[" returns the value for the specified lag. The index should be a single number. This is the recommended way to extract the value at a single index.

```
signature(x = "FlexibleLagged", i = "ANY", j = "ANY")
signature(x = "Lagged", i = "numeric", j = "missing")
signature(x = "Lagged2d", i = "numeric", j = "logical")
signature(x = "Lagged2d", i = "numeric", j = "numeric")
signature(x = "slMatrix", i = "numeric", j = "ANY")
signature(x = "Lagged2d", i = "missing", j = "numeric")
signature(x = "Lagged2d", i = "numeric", j = "missing")
signature(x = "FlexibleLagged", i = "missing", j = "numeric")
signature(x = "FlexibleLagged", i = "numeric", j = "missing")
```

Examples

```
## for 1d objects the difference of '[' and '[' is not visible
(acv1 <- acf2Lagged(acf(1deaths, plot = FALSE)))
acv1[1]
acv1[[1]]

## in higher dimensions it matters
acv2 <- acf2Lagged(acf(ts.union(mdeaths, fdeaths), plot = FALSE))
acv2[0] # array
acv2[[0]] # matrix

## as in standard R conventions, '[' can select arbitrary number of elements
acv2[0:1]
## ... while '[' selects only one, so this is an error:
## Not run:
acv2[[0:1]]

## End(Not run)

## Lagged2
m <- matrix(10:49, nrow = 4, byrow = TRUE)
m_lagged <- Lagged(m)
m_lagged
```

```
## one index: lag
m_lagged[0:1]
m_lagged[0] # column vector
m_lagged[[0]] # numeric
## two indices: first is row, second is lag
m_lagged[1, 0] # '[' doesn't drop dimensions
m_lagged[1, 0:3]

m_lagged[[1, 0]] # '[' does drop dimensions
m_lagged[[1, 0:3]]
m_lagged[[1, TRUE]] # the whole first row, as numeric

m_lagged[1:2, 0:3] # ok, a matrix
## ... but the first arg. of '[' must be of length one,
## so this throws error:
## Not run:
m_lagged[[1:2, 0:3]]

## End(Not run)
```

[\[\[<-methods](#)*Methods for '['<- in package 'lagged'*

Description

Methods for '['<- in package 'lagged'.

Methods

signature(x = "Lagged", i = "numeric")

See Also

[\[\[<-methods](#)

Index

- * **acf**
 - sl2acfbase, [17](#)
- * **classes**
 - FlexibleLagged-class, [4](#)
 - Lagged-class, [9](#)
 - Lagged1d-class, [10](#)
 - Lagged2d-class, [11](#)
 - Lagged3d-class, [12](#)
 - slMatrix-class, [20](#)
- * **methods**
 - [~methods, [21](#)
 - [<~methods, [22](#)
 - [[~methods, [23](#)
 - [[<~methods, [24](#)
 - maxLag, [13](#)
 - maxLag<~, [15](#)
 - nSeasons, [16](#)
- * **programming**
 - Lagged, [6](#)
- * **seasonLag**
 - slMatrix, [18](#)
- * **ts**
 - acf2Lagged, [2](#)
 - Lagged, [6](#)
 - maxLag, [13](#)
 - maxLag<~, [15](#)
 - nSeasons, [16](#)
- [, ANY, ANY, ANY, ANY-method ([~methods), [21](#)
- [, FlexibleLagged, missing, ANY, ANY-method ([~methods), [21](#)
- [, FlexibleLagged, missing, missing, ANY-method ([~methods), [21](#)
- [, FlexibleLagged, numeric, missing, logical-method ([~methods), [21](#)
- [, FlexibleLagged, numeric, missing, missing-method ([~methods), [21](#)
- [, Lagged, missing, ANY, ANY-method ([~methods), [21](#)
- [, Lagged1d, numeric, ANY, ANY-method ([~methods), [21](#)
- [, Lagged2d, ANY, ANY, character-method ([~methods), [21](#)
- [, Lagged2d, character, character, missing-method ([~methods), [21](#)
- [, Lagged2d, character, missing, logical-method ([~methods), [21](#)
- [, Lagged2d, character, missing, missing-method ([~methods), [21](#)
- [, Lagged2d, character, numeric, missing-method ([~methods), [21](#)
- [, Lagged2d, missing, character, missing-method ([~methods), [21](#)
- [, Lagged2d, missing, numeric, missing-method ([~methods), [21](#)
- [, Lagged2d, numeric, character, missing-method ([~methods), [21](#)
- [, Lagged2d, numeric, missing, logical-method ([~methods), [21](#)
- [, Lagged2d, numeric, missing, missing-method ([~methods), [21](#)
- [, Lagged2d, numeric, numeric, missing-method ([~methods), [21](#)
- [, Lagged3d, numeric, missing, logical-method ([~methods), [21](#)
- [, Lagged3d, numeric, missing, missing-method ([~methods), [21](#)
- [, nonStructure, ANY, ANY, ANY-method ([~methods), [21](#)
- [, slMatrix, ANY, ANY, ANY-method ([~methods), [21](#)
- [~methods, [21](#)
- [<~methods, [22](#)
- [<~, FlexibleLagged, missing-method ([<~methods), [22](#)
- [<~, FlexibleLagged, numeric-method ([<~methods), [22](#)
- [<~, Lagged, missing-method ([<~methods), [22](#)

- [<-, Lagged1d, numeric-method
([<-methods), 22
- [<-, Lagged2d, numeric-method
([<-methods), 22
- [<-, Lagged3d, numeric-method
([<-methods), 22
- [<-, slMatrix, ANY-method ([<-methods), 22
- [[, FlexibleLagged, ANY, ANY-method
([[methods), 23
- [[, FlexibleLagged, missing, numeric-method
([[methods), 23
- [[, FlexibleLagged, numeric, missing-method
([[methods), 23
- [[, Lagged, numeric, missing-method
([[methods), 23
- [[, Lagged2d, missing, numeric-method
([[methods), 23
- [[, Lagged2d, numeric, logical-method
([[methods), 23
- [[, Lagged2d, numeric, missing-method
([[methods), 23
- [[, Lagged2d, numeric, numeric-method
([[methods), 23
- [[, slMatrix, numeric, ANY-method
([[methods), 23
- [[methods, 23
- [[<-methods, 24
- [[<-, Lagged, numeric-method
([[<-methods), 24
- acf2Lagged, 2, 18
- acfbase2sl (sl2acfbase), 17
- dataWithLagNames, 3
- fill.slMatrix, 19
- FlexibleLagged-class, 4
- Lagged, 4, 5, 6, 10–13
- Lagged-class, 9
- Lagged1d, 5, 6, 10, 12, 13
- Lagged1d-class, 10
- Lagged2d, 5, 6, 10, 11, 13
- Lagged2d-class, 11
- Lagged3d, 5, 6, 10–12
- Lagged3d-class, 12
- maxLag, 13, 15
- maxLag, ANY-method (maxLag), 13
- maxLag, array-method (maxLag), 13
- maxLag, Lagged-method (maxLag), 13
- maxLag, matrix-method (maxLag), 13
- maxLag, slMatrix-method (maxLag), 13
- maxLag, vector-method (maxLag), 13
- maxLag-methods (maxLag), 13
- maxLag<-, 15
- maxLag<-, FlexibleLagged-method
(maxLag<-), 15
- maxLag<-, Lagged-method (maxLag<-), 15
- maxLag<--methods (maxLag<-), 15
- nSeasons, 16
- nSeasons, slMatrix-method (nSeasons), 16
- nSeasons-methods (nSeasons), 16
- nSeasons<- (nSeasons), 16
- sl2acfbase, 17
- sl2vecacf (sl2acfbase), 17
- slMatrix, 18, 21
- slMatrix-class, 20