Package: mcompanion (via r-universe)

August 21, 2024

Type Package

Title Objects and Methods for Multi-Companion Matrices

Version 0.5.8.9000

Description Provides a class for multi-companion matrices with methods for arithmetic and factorization. A method for generation of multi-companion matrices with prespecified spectral properties is provided, as well as some utilities for periodically correlated and multivariate time series models. See Boshnakov (2002) <doi:10.1016/S0024-3795(01)00475-X> and Boshnakov & Iqelan (2009) <doi:10.1111/j.1467-9892.2009.00617.x>.

URL https://geobosh.github.io/mcompanion/ (doc), https://github.com/GeoBosh/mcompanion (devel)

BugReports https://github.com/GeoBosh/mcompanion/issues

Depends methods

Imports Matrix (>= 1.5-0), gbutils, MASS, Rdpack

Suggests testthat, lagged

RdMacros Rdpack

License GPL (>= 2)

Collate mc.R mcompanion.R utils_Jordan.R mat.R sim.R class_MC.R class_MF.R class_Jordan.R chains_smc.R class_SMC.R class_mcSpec.R

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

RemoteUrl https://github.com/geobosh/mcompanion

RemoteRef HEAD

RemoteSha 734b9ddd31270b5eea1ff9c850de82b7bd013146

Contents

`mcompanion-package` *Objects and Methods for Multi-Companion Matrices*

Description

Provides a class for multi-companion matrices with methods for arithmetic and factorization. A method for generation of multi-companion matrices with prespecified spectral properties is provided, as well as some utilities for periodically correlated and multivariate time series models. See Boshnakov (2002) <doi:10.1016/S0024-3795(01)00475-X> and Boshnakov & Iqelan (2009) <doi:10.1111/j.1467-9892.2009.00617.x>.

Details

Index of the main exported objects, classes and methods:

Classes and generators:

MultiCompanion-class	Class "MultiCompanion"
MultiFilter-class	Class "MultiFilter"
VAR2pcfilter	PAR representations of VAR models
mCompanion	Create objects from class MultiCompanion
mcSpec	Generate objects of class mcSpec
mcSpec-class	A class for spectral specifications of multi-companion matrices
mf_VSform	Extract properties of multi-filters

Utilities for multi-companion matrices:

mc_eigen	The eigen decomposition of a multi-companion matrix
mc_factorize	Factorise multi-companion matrices
mc_factors	Factors of multi-companion matrices
mc_from_factors	Multi-companion matrix from factors

Simulation:

sim_mc	Simulate a multi-companion matrix
sim_pcfilter	Generate periodic filters

Generic matrix utilities:

Jordan_matrix	Utilities for Jordan matrices
mcStable	Check if an object is stable
rblockmult	Right-multiply a matrix by a block

Spectral description of mc-matrices:

spec_core	Parameterise Jordan chains of multi-companion matrices
spec_root0	Give the spectral parameters for zero eigenvalues of mc-matrices
spec_root1	Give the spectral parameters for eigenvalues of mc-matrices equal to one
spec_seeds1	Generate seed parameters for unit mc-eigenvectors

Low-level functions:

mc_chain_extend	Extend multi-companion eigenvectors

Overview of the package

Package "mcompanion" implements multi-companion matrices as discussed by Boshnakov (2002) and Boshnakov and Iqelan (2009). The main feature is the provided parsimonious parameterisation of such matrices based on their eigenvalues and the seeds for their eigenvectors. This can be used for specification and parameterisation of models for time series and dynamical systems in terms of spectral characteristics, such as the poles of the associated filters or transition matrices.

A multi-companion matrix of order k is a square $n \times n$ matrix with arbitrary k rows put on top of an identity $(n-k) \times (n-k)$ matrix and a zero $(n-k) \times k$ matrix. The number k is the multi-companion order of the matrix. It may happen that the top $k \times n$ block, say T, of an mc-matrix has

columns of zeroes at its end. In this documentation we say that an $n \times n$ matrix has dimension n and size $n \times n$.

Multi-companion matrices can be created by the functions `new` and `mCompanion`, the latter being more versatile. Some of the other functions in this package return such objects, as well.

`sim_mc` generates a multi-companion matrix with partially or fully specified spectral properties. If the specification is incomplete, it completes it with simulated values.

`sim_pcfilter` is a convenience function (it uses `sim_mc`) for generation of filters for periodically correlated models. These can be converted to various multivariate models, such as VAR, most conveniently using class `MultiFilter`, see below.

Class "MultiFilter" is a formal representation of periodic filters with methods for conversion between periodic and (non-periodic) multivariate filters. Several forms of VAR models are provided, see `mf_VSform`, `VAR2pcfilter`, `MultiFilter`, and the examples there.

Author(s)

Georgi N. Boshnakov [aut, cre]

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

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN (2007). "Singular value decomposition of multi-companion matrices." *Linear Algebra Appl.*, **424**(2-3), 393–404. ISSN 0024-3795, doi:10.1016/j.laa.2007.02.010.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

for examples, see `mCompanion`, `sim_mc`, `sim_pcfilter`, `mf_VSform`, `VAR2pcfilter`, `MultiFilter`, `MultiCompanion`,

Examples

```
## see the examples in the links in section 'See Also' above.
```

`jordan` *Utilities for Jordan matrices*

Description

Utility functions for work with the Jordan decompositions of matrices: create a block diagonal matrix of Jordan blocks, restore a matrix from its Jordan decomposition, locate specific chains.

Usage

```
Jordan_matrix(eigval, len.block)
from_Jordan(x, jmat, ...)
chain_ind(chainno, len.block)
chains_to_list(vectors, heights)
```

Arguments

eigval	eigenvalues, a numeric or complex vector.
len.block	lengths of Jordan chains, a vector of positive integers.
x	generalised eigenvectors, a matrix with one column for each (generalised) eigenvector.
jmat	a Jordan matrix.
chainno	a vector of positive integers between 1 and length(eigval) specifying which Jordan chains to locate, see Details.
...	further arguments to pass on to solve.
vectors	a matrix of generalised eigenvectors of a matrix.
heights	a vector of chain lengths, heights[i] is the length of the i-th chain.

Details

Jordan_matrix creates a Jordan matrix (block-diagonal matrix with Jordan blocks on the diagonal) whose i-th diagonal block corresponds to eigval[i] and is of size len.block[i]. If len.block is missing, Jordan_matrix returns diag(eigenvalues).

from_Jordan computes the matrix whose Jordan decomposition is represented by arguments X (chains) and J (Jordan matrix). Conceptually, the result is equivalent to XJX^{-1} but without explicitly inverting matrices (currently the result is the transpose of solve(t(x), t(x %*% jmat), ...)).

chain_ind computes the columns of specified Jordan chains in a matrix of generalised eigenvectors. It is mostly internal function. If x is a matrix whose columns are generalised eigenvectors and the i-th Jordan chain is of length len.block[i], then this function gives the column numbers of x containing the specified chains. Note that chain_ind is not able to deduce the total number of eigenvalues. It is therefore an error to omit argument len.block when calling it.

chains_to_list converts the matrix vectors into a list of matrices. The i-th element of this list is a matrix whose columns are the vectors in the i-th chain.

Value

for Jordan_matrix, a matrix with the specified Jordan blocks on its diagonal.

for from_Jordan, the matrix with the specified Jordan decomposition.

for chain_ind, a vector of positive integers giving the columns of the requested chains.

for chains_to_list, a list of matrices.

Level

0

Author(s)

Georgi N. Boshnakov

Examples

```
## single Jordan blocks
Jordan_matrix(4, 2)
Jordan_matrix(5, 3)
Jordan_matrix(6, 1)
## a matrix with the above 3 blocks
Jordan_matrix(c(4, 5, 6), c(2, 3, 1))

## a matrix with a 2x2 Jordan block for eval 1 and two simple 0 eval's
m <- make_mcmatrix(eigval = c(1), co = cbind(c(1,1,1,1), c(0,1,0,0)),
                     dim = 4, len.block = c(2))
m
m.X <- cbind(c(1,1,1,1), c(0,1,0,0), c(0,0,1,0), c(0,0,0,1))
m.X
m.J <- cbind(c(1,0,0,0), c(1,1,0,0), rep(0,4), rep(0,4))
m.J

from_Jordan(m.X, m.J)          # == m
m.X %*% m.J %*% solve(m.X) # == m
all(m == from_Jordan(m.X, m.J)) && all(m == m.X %*% m.J %*% solve(m.X))
## TRUE

## which column(s) in m.X correspond to 1st Jordan block?
chain_ind(1, c(2,1,1)) # c(1, 2) since 2x2 Jordan block

## which column(s) in m.X correspond to 2nd Jordan block?
chain_ind(2, c(2,1,1)) # 3, simple eval

## which column(s) in m.X correspond to 1st and 2nd Jordan blocks?
chain_ind(c(1, 2), c(2,1,1)) # c(1,2,3)
## non-contiguous subset are ok:
chain_ind(c(1, 3), c(2,1,1)) # c(1,2,4)

## split the chains into a list of matrices
chains_to_list(m.X, c(2,1,1))

m.X %*% m.J
m %*% m.X    # same
all(m.X %*% m.J == m %*% m.X)    # TRUE

m %*% c(1,1,1,1)    # = c(1,1,1,1),  evec for eigenvalue 1
m %*% c(0,1,0,0)    # gen.e.v. for eigenvalue 1
## indeed:
all( m %*% c(0,1,0,0) == c(0,1,0,0) + c(1,1,1,1) ) # TRUE

## m X = X jordan.block
cbind(c(1,1,1,1), c(0,1,0,0)) %*% cbind(c(1,0), c(1,1))
m %*% cbind(c(1,1,1,1), c(0,1,0,0))
```

`JordanDecomposition` *Create objects representing Jordan decompositions*

Description

Create objects representing Jordan decompositions.

Usage

```
JordanDecomposition(values, vectors, heights, ...)
```

Arguments

`values`	eigenvalues, a vector of length equal to the number of Jordan chains.
`vectors`	the (generalised) eigenvectors, a matrix.
`heights`	a vector of positive integers, `heights[i]` is the height of `values[i]`.
`...`	further arguments that may be needed by methods.

Details

`JordanDecomposition` is an S4 generic function. It creates objects representing Jordan decompositions. Dispatch is on the first two arguments, `values` and `vectors`.

The names of the arguments correspond to slots in class "JordanDecompositionDefault", which is the class of the objects created by methods in package **mcompanion** and inherits from the virtual class "JordanDecomposition".

Value

an object inheriting from "JordanDecomposition"

Methods

`signature(values = "ANY", vectors = "ANY")` the default method; currently raises an error.

`signature(values = "JordanDecomposition", vectors = "missing")` simply returns `values`.

`signature(values = "list", vectors = "missing")` In this case `values` can be a list with components `"values"`, `"vectors"` and `"heights"`. This method has an additional argument `"names"` which can be used when the components of the list are different, e.g.
`names = c(values = "eigval", vectors = "eigvec", heights = "len.block")`.

`signature(values = "missing", vectors = "matrix")` This is equivalent to the case `values = "number"` with `values` set to a vector of missing values.

`signature(values = "missing", vectors = "missing")` `values` (`vectors`) is set to a vector (matrix) of missing values. The dimensions are deduced from argument `heights`, so `heights` cannot be missing for this signature.

`signature(values = "number", vectors = "matrix")` This is equivalent to calling `new` for class `"JordanDecompositionDefault"` with arguments `values`, `vectors` and `heights`.

`signature(values = "number", vectors = "missing")` This is equivalent to the case `vectors = "matrix"` with `vectors` set to a matrix of missing values.

`signature(values = "SmallMultiCompanion", vectors = "missing")` This computes the Jordan decomposition of an object from class "SmallMultiCompanion".

Author(s)

Georgi N. Boshnakov

Examples

```
m <- matrix(c(1,2,4,10), nrow = 2)
m <- matrix(c(1,2,4,10), nrow = 2)
m <- matrix(c(5, 12, 3, 4), nrow = 2)

JordanDecomposition(values = rep(0,2), vectors = m)
jd <- JordanDecomposition(values = c(0.9, 0.3), vectors = m)
as(jd, "matrix")
eigen(jd)
## the eigenvectors are scaled versions of m's columns:
eigen(jd)$vectors %*% diag(c(5 / eigen(jd)$vectors[1,1], -5))
## == m

## eigenvalues are not supplied, so set to  NA's here:
JordanDecomposition(vectors = m)

## eigenvectors are set to vectors of NA's here:
JordanDecomposition(values = rep(0,2), height = c(1,1))
```

JordanDecompositionDefault-class

A basic class for Jordan decompositions

Description

A basic class for Jordan decompositions.

Details

Class `"JordanDecompositionDefault"` represents Jordan decompositions. It inherits from the virtual class "JordanDecomposition", which serves as a base class for Jordan decompositions. These classes should be considered internal.

Objects from the Class

Objects from class `"JordanDecompositionDefault"` can be created by a call to `JordanDecomposition()`.

Objects can be created by calls of the form `new("JordanDecompositionDefault", heights, ...)`.

Slots

`values`: Object of class `"number"`, vector of eigenvalues (one value for each Jordan chain).

`heights`: Object of class `"integer"`, the heights of the Jordan chains.

`vectors`: Object of class `"matrix"`, the (generalised) eigenvectors (similarity matrix).

Extends

Class `"JordanDecomposition"`, directly.

Methods

coerce `signature(from = "JordanDecompositionDefault", to = "matrix")`:
gives the matrix represented by the Jordan decomposition, i.e. XJX^{-1}. As with other coerce methods, use `as(obj, "matrix")`, where `obj` is the Jordan decomposition object.

initialize `signature(.Object = "JordanDecompositionDefault")`: ...

Author(s)

Georgi N. Boshnakov

See Also

`JordanDecomposition`

Examples

```
showClass("JordanDecompositionDefault")

m <- matrix(c(1,2,4,3), nrow = 2)
new("JordanDecompositionDefault", values = rep(0,2), vectors = m)
```

---

make_mcev *Create a multi-companion eigenvector*

Description

Creates an eigenvector of a multicompanion matrix from the eigenvalue and the seed parameters.

Usage

```
make_mcev(eigval, co, dim, what.co = "bottom")
make_mcgev(eigval, co, v, what.co = "bottom")
```

Arguments

eigval	the eigenvalue.
co	the bottom (default) or the top seed elements of the vector.
dim	the size of the matrix.
what.co	type of co: "bottom" or "top".
v	the previous vector in the chain.

Details

make_mcev computes an eigenvector for a multi-companion dim x dim matrix by filling its top or bottom part with co and completing the remaining elements using the general pattern of eigenvectors of such matrices (Boshnakov 2002).

Similarly, make_mcgev computes the next generalised eigenvector in a chain whose previous element is v.

what.co cannot be "top" if the eigenvalue is 0. Generalised eigenvectors corresponding to the zero eigenvalue have some specifics, so it is better to use the specialised functions in that case.

Value

make_mcev returns the required eigenvector.

make_mcgev returns the required generalised eigenvector.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Examples

```
v1 <- make_mcev(0.5, c(1, 1), dim = 4)
v1
make_mcev(0.5, c(1, 1), dim = 4, what.co = "top")

v2 <- make_mcgev(0.5, c(0, 1), v = v1, what.co = "top")
v2
make_mcgev(0.5, c(0, 1), v = v2, what.co = "top")
```

make_mcmatrix *Generate a multi-companion matrix from spectral description*

Description

Generate a multi-companion matrix or its Jordan decomposition from spectral parameters.

Usage

```
make_mcmatrix(type = "real", what.res = "matrix", ..., eigval0)

make_mcchains(eigval, co, dim, len.block, eigval0 = FALSE,
              mo.col = NULL, what.co = "bottom", ...)
```

Arguments

eigval	the eigenvalues, a numeric vector
co	the seeding parameters for the eigenvectors, a matrix
dim	the dimension of the matrix, a positive integer
len.block	lengths of Jordan chains, len.block[i] is for eigval[i]
type	mode of the matrix, real or complex
what.res	format of the result, see details
eigval0	If TRUE completes the matrix to a square matrix, see details. eigval0 is ignored by make_mcmatrix (it always sets it to TRUE).
...	for make_mcmatrix, these are additional arguments to be passed to make_mcchains. For make_mcchains, arguments in "..." are passed on to mc_0chains.
mo.col	the last non-zero column in the top of the mc-matrix. The default is dim.
what.co	a character string equal to "bottom" (default) or "top", specifying whether the 'co' parameters give the last or the first few elements of the (generalised) eigenvectors.

Details

make_mcmatrix creates a multi-companion matrix specified by spectral parameters. make_mcchains creates a matrix of eigenvectors and generalised eigenvectors from the given spectral parameters.

make_mcmatrix passes the spectral parameters to make_mcchains to generate the (generalised) eigenvectors. It then calls Jordan_matrix to create the corresponding Jordan matrix. The results are combined to produce the multicompanion matrix. By default, the real part is returned, which is appropriate if all complex spectral parameters come in complex conjugate pairs. This may be changed by argument type. A list containing the matrix and the Jordan factors is returned if what.res = "list".

The closely related function sim_mc is like make_mcmatrix but it does not need complete specification of the matrix - it completes any missing information (eigenvalues, co) with randomly generated entries. The result of both functions is a list or ordinary matrix, use mCompanion to obtain a MultiCompanion object directly.

`make_mcchains` constructs the eigensystem, `make_mcmatrix` calls `make_mcchains` (passing the `...` arguments to it) and forms the matrix. `make_mcchains` passes the `...` arguments to `mc_0chains`.

`make_mcchains` creates the full eigenvectors from the `co` parameters. If the number of vectors is smaller then `dim` and `eigval0` is TRUE it then completes the system with chains for the zero eigenvalue. More specifically, it assumes that the number of the given chains is `mo.col`, takes chains corresponding to the zero eigenvalue, if any, and adds additional eigenvectors and/or generalised eigenvectors to construct the complete system.

The mc-order is determined from the dimension of the 'co' parameters. If that is equal to `dim`, the mc-matrix is actually a general matrix.

TODO: cover the case `mo` < `mo.col`?

Value

`make_mcmatrix` normally returns the multi-companion matrix (as an ordinary matrix) having the given spectral properties but if `what.res` = `"list"`, it returns a list containing the matrix and the spectral information:

`eigval`	eigenvalues, a vector
`len.block`	lengths of Jordan chains, a vector
`mo`	multi-companion order, positive integer
`eigvec`	generalied eigenvectors, a matrix
`co`	seeding parameters
`mo.col`	top order
`mat`	the multi-companion matrix, a matrix

`make_mcchains` returns a similar list without the component `mat`.

Note

The result is an ordinary matrix. Also, some entries that should be 0 may be non-zero due to numerical error.

To get a MultiCompanion object use `mCompanion`.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

`make_mcev`, `make_mcgev`, `sim_mc`, `mCompanion`

Examples

```
make_mcmatrix(eigval = c(1, 0.5), co = cbind(c(1,1), c(1, -1)), dim = 4,
              mo.col = 2,
              len.block = c(1, 1))

## one unit root, one root = 0.5
make_mcmatrix(eigval = c(1, 0.5), co = cbind(c(1,1), c(1, -1)), dim = 6,
              mo.col = 2,
              len.block = c(1, 1))

## two simple unit roots, one root = 0.5
make_mcmatrix(eigval = c(1, 1, 0.5), co = cbind(c(1,1), c(1, -1), c(1, 1)), dim = 6,
              mo.col = 3,
              len.block = c(1, 1, 1))

## two unit roots with a single Jordan chain, one root = 0.5
make_mcmatrix(eigval = c(1, 0.5), co = cbind(c(1,1), c(0, 1), c(1, 1)), dim = 6,
              len.block = c(2, 1))


## make_mcchains
make_mcchains(c(1, 0.5), co = cbind(c(1,1), c(1, 1)), dim = 4,
              len.block = c(1, 1), eigval0 = TRUE)

## one unit root, one root = 0.5
make_mcchains(c(1, 0.5), co = cbind(c(1,1), c(1, 1)), dim = 6,
              len.block = c(1, 1), eigval0 = TRUE)

## two simple unit roots, one root = 0.5
make_mcchains(c(1, 1, 0.5), co = cbind(c(1,1), c(1, -1), c(1, 1)), dim = 6,
              len.block = c(1, 1, 1), eigval0 = TRUE)

## two unit roots with a single Jordan chain, one root = 0.5
make_mcchains(c(1, 0.5), co = cbind(c(1,1), c(1, -1), c(1, 1)), dim = 6,
              len.block = c(2, 1), eigval0 = TRUE)



## examples with mc-order = dim
make_mcchains(c(1), co = cbind(c(1,1,1,1), c(1,2,1,1)), dim = 4,
              len.block = c(2), eigval0 = TRUE)
## do not complete with chians for the 0 eigval:
make_mcchains(c(1), co = cbind(c(1,1,1,1), c(1,2,1,1)), dim = 4,
              len.block = c(2), eigval0 = FALSE)

make_mcmatrix(eigval = c(1), co = cbind(c(1,1,1,1), c(1,2,1,1)), dim = 4,
              len.block = c(2))
make_mcmatrix(eigval = c(1), co = cbind(c(1,1,1,1), c(1,2,3,4)), dim = 4,
              len.block = c(2))
```

`mCompanion` *Create objects from class MultiCompanion*

Description

Create, generate, or simulate objects from class `"MultiCompanion"` by specifying the matrix in several ways.

Usage

```
mCompanion(x, detect = "nothing", misc = list(), ...)

## S4 method for signature 'MultiCompanion'
initialize(.Object, xtop, mo, n, mo.col, ido, x, dimnames,
           detect = "nothing", misc = list())
```

Arguments

`x`	the matrix or, for `mCompanion` only, the top of the matrix or a character string, see section 'Details'.
`misc`	information to be stored in the object's pad.
`...`	other arguments to be passed down to generator functions, see section 'Details'.
`xtop`	the top of the matrix.
`mo`	the multi-companion order of the matrix.
`n`	the dimension.
`mo.col`	the top order, meaniing that columns mo.col+1,...,n of the top of the matrix are zeros. `mo.col` may also be set to "detect", in which case it is determined by scanning `xtop` or `x`.
`ido`	the dimension of the identity sub-matrix.
`dimnames`	is not used currently.
`detect`	controls whether automatic detection of `mo` and `mo.col` should be attempted. The values tested are "mo", "mo.col", "all", and "nothing" with obvious meanings.
`.Object`	this is set implicitly by package "methods".

Details

Objects from class `"MultiCompanion"` can be created by calling `mCompanion()` or `new("MultiCompanion", ...)`. In the latter case the "`...`" arguments are as for the `initialize` method, except `.Object`. Do not call `initialize` directly.

`mCompanion` can generate multi-companion matrices from spectral information, full or partial, using the methodology developed by Boshnakov and Iqelan (2009). If the specification is not given in full, the missing information is filled with suitably simulated values. For example, unspecifies eigenvalues are generated inside the unit circle, `sim_mc`.

If argument x is the string "sim" or "gen", then mCompanion calls sim_mc or make_mcmatrix, respectively, with the arguments ... and converts the result to class MultiCompanion. See the documentation of those functions for further details and examples. The conversion may be the main reason to use mCompanion in this way rather than call sim_mc and make_mcmatrix directly.

Otherwise, if x is numeric it is taken to specify the top of the matrix unless detect="mo" in which case it is the whole matrix. In both cases all arguments are passed down to new, the only (more or less) change being that x is passed down as xtop=x and x=x, respectively, see MultiCompanion.

detect=="gen" signifies that x has the format of the output from sim_mc or make_mcmatrix, so that mCompanion may use the additional information in such objects.

The multi-companion order is determined automatically from the content of the matrix if detect=="mo".

Value

a multi-companion matrix, an object of class "MultiCompanion"

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN (2007). "Singular value decomposition of multi-companion matrices." *Linear Algebra Appl.*, **424**(2-3), 393–404. ISSN 0024-3795, doi:10.1016/j.laa.2007.02.010.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

sim_mc, make_mcmatrix, MultiCompanion

Examples

```
# simulate a 6x6 mc matrix with 2 non-trivial rows
mCompanion("sim", dim = 4, mo = 2)
# simulate a 6x6 mc matrix with 4 non-trivial rows
mCompanion("sim", dim = 6, mo = 4)
# similar to above but top rows with 2 non-zero columns
mCompanion("sim", dim = 6, mo = 4, mo.col = 2)

## specify the non-trivial top rows (as a matrix):
m1 <- matrix(1:24, nrow = 4)
mCompanion(m1)            # mc matrix with m1 on top

m2 <- rbind(c(1, 2, 0, 0), c(3, 4, 0, 0))
x2a <- mCompanion(m2)     # mc matrix with m2 on top
x2a@mo.col                #  = 4
```

```
x2 <- mCompanion(m2, mo.col = "detect")
x2@mo.col                #  = 2,  detects the 0 columns in m2
mCompanion(m2, mo.col = 2)  # same

# create manually an mc matrix
(m3 <- rbind(m1, c(1, rep(0, 5)), c(0, 1, rep(0, 4))))
# turn it into a MultiCompanion object
x3 <- mCompanion(x = m3, detect = "mo")
x3@mo
x3 <- mCompanion(m3)
x3@mo

m4 <- rbind(c(1, 2, rep(0, 4)), c(3, 4, rep(0, 4)))

x4 <- mCompanion(m4, mo = 2)
x4@mo.col    #  = 6,
## special structure not incorporated in x4,
##   eigen and mc_eigen are equiv. in this case
eigen(x4)
mc_eigen(x4)

x4a <- mCompanion(m4, mo = 2, mo.col = 2)
x4a@mo.col    #  = 2,  has Jordan blocks of size > 1
## the eigenvectors do not span the space:
eigen(x4a)
## mc_eigen exploits the Jordan structure, e.g.2x2 Jordan blocks,
## and gives the generalised eigenvectors:
(ev <- mc_eigen(x4a))

x4a %*% ev$vectors

## construct the Jordan matrix of x4a from eigenvalues and eigenvectors
(x4a.j <- Jordan_matrix(ev$values,  ev$len.block))

## check that AX = XJ and A = XJX^-1, up to numerical precision:
x4a %*% ev$vectors  -  ev$vectors %*% x4a.j
x4a  -  ev$vectors %*% x4a.j %*% solve(ev$vectors)
```

---

mcSpec *Generate objects of class mcSpec*

Description

Generate objects of class mcSpec.

Usage

```
mcSpec(...)

## S4 method for signature 'mcSpec'
```

```
initialize(.Object, dim, mo, root1 = numeric(0), iorder = 0,
           siorder = 0, order = rep(dim, mo), evtypes = NULL,
           mo.col = NULL, n.roots = mo.col, ...)
```

Arguments

dim	the dimension, a positive integer.
mo	multi-companion order, a.k.a. number of seasons.
root1	roots equal to one, a vector of positive integers of length at most mo.
iorder	integration order, a non-negative integer.
siorder	seasonal integration order, a non-negative integer.
order	order of the periodic filter, a vector of length mo.
evtypes	types of additional eigenvalues, see Details.
mo.col	number of non-zero columns in the top part of the multicompanion matrix, see Details.
n.roots	number of non-zero roots
...	further arguments to be passed on.
.Object	An object. This argument is not used in calls of mcSpec and new, see the details section.

Details

mcSpec(...) and new("mcSpec", ...) create objects from class mcSpec. The two calls are equivalent and may contain any of the arguments of the initialize method described here, except .Object which is generated automatically. In both cases the initialize method is called and passed all the arguments.

Several ways are provided for the specification of unit roots and they may be combined, as long as the specification is consistent.

roots1 specifies eigenvalues equal to 1 and the size of their Jordan chains. iorder and siorder provide convenient shortcuts for the special cases which they cover.

iorder specifies the integration order. This corresponds to operator $(1 - B)$ applied iorder times.

Similarly, siorder specifies the seasonal integration order, which corresponds to the operator $(1 - B^s)$ applied siorder times, where s is equal to mo. This argument generates mo unit roots, each of height (dimension of its Jordan chain) siorder.

It is possible to use combinations of these arguments to specify the unit roots and all specifications are combined. Care must be taken not to exceed dim.

If mo.col is missing, it is set to max(order). mo.col may also be the character string "+ones". In this case the dimension of the unit roots is added to max(order). mo.col may also be set directly by giving it an appropriate integer value. **TODO: Need more checks for consistency here!**

TODO: describe other roots and eigenvectors!

After all specified quantities are prepared, the rest are set to NA's.

If not all eigenvalues are specified, additional eigenvalues are introduced to reach dimension dim. By default, if an even number of eigenvalues is needed, all of them are specified as complex pairs, "cp". If the number is odd, one real eigenvalue is specified and the rest are set again to "cp".

Argument `evtypes` can be used to select a different setting for the additional eigenvalues. It is a character vector in which "r" stands for real eigenavalues and "cp" stands for a complex pair. For example, if there are two "free" eigenvalues, the automatic choice would be a complex pair, "cp". If two real eigenvalues are desired set `evtypes` to `c("r","r")`.

Note: `evtypes` is for types of additional eigenvalues. Do not specify types for eigenvalues equal to one or zero.

Value

an object of class `mcSpec`

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

`mcSpec-class`

Examples

```
spec2 <- mcSpec(21, 4, siorder=2, iorder=1)
spec4 <- mcSpec(11, 4, siorder=1, iorder=1)
spec.co2 <- mcSpec(dim = 5, mo = 4, siorder = 1)
spec.co2new <- mcSpec(dim = 5, mo = 4, siorder = 1) # after correcting ev.arg
spec.co2alt <- mcSpec(dim = 6, mo = 4, siorder = 1)

spec.co3 <- mcSpec(dim = 5, mo = 4, root1 = c(1,1,1))

spec.coz1 <- mcSpec(dim = 4, mo = 4, root1 = c(1,1), order = rep(2,4)) # test0 roots
spec.coz2 <- mcSpec(dim = 5, mo = 4, root1 = c(1,1), order = rep(2,4)) # test0 roots
spec.coz3 <- mcSpec(dim = 4, mo = 4, root1 = c(1), order = rep(2,4)) # test0 roots
spec.co4 <- mcSpec(dim = 4, mo = 4, root1 = c(1,1,1))
```

`mcSpec-class` *A class for spectral specifications of multi-companion matrices*

Description

A class for spectral specifications of multi-companion matrices.

Objects from the Class

Objects can be created by calls of one of the following equivalent forms:

- `mcSpec(dim, mo, root1, iorder, siorder, order, evtypes, ...)`,
- `new("mcSpec", dim, mo, root1, iorder, siorder, order, evtypes, ...)`.

An object of class "mcSpec" holds a spectral specification of a square multi-companion matrix. The specification may be only partial. In that case unspecified components are set to `NA`.

Eigenvalues are represented by their modulus and complex argument. The argument is in cycles per unit time. So, a negative real number has argument 0.5.

The complex eigenvalues come in pairs and only one needs to be specified. If an eigenvalue is not simple, it should not be repeated. Rather, the size of the corresponding Jordan block should be specified.

The types of the eigenvalues may be "r" (real) or "cp" (complex pair).

See `mcSpec` for full details about the initialization function for class `mcSpec`.

Slots

`dim`**:** dimension of the matrix, a positive integer.

`mo`**:** multi-companion order, a positive integer.

`ev.type`**:** Types of eigenvalues, "r" or "cp", a character vector.

`co.type`**:** Types of the `co` parameters, a character vector.

`order`**:** orders of the factors, the default is `rep(dim,mo)`.

`n.root`**:** number of nonzero roots.

`ev.abs`**:** absolute values (moduli) of the roots.

`ev.arg`**:** complex arguments of the roots (cycles per unit time). In particular, zero for positive reals, 0.5 for negative reals. (**TODO:** check that functions that use this specification know that!)

`block.length`**:** sizes of Jordan blocks corresponding to the eigenvalues, a vector of positive integers. By default the eigenvalues are simple.

`co.abs`**:** moduli of the `co` parameters, a matrix.

`co.arg`**:** arguments of the `co` parameters, a matrix.

`mo.col`**:** Object of class `"numeric"`.

`F0bot`**:** Object of class `"optionalMatrix"`.

Methods

initialize `signature(.Object = "mcSpec")`: see `mcSpec`.

Note

The initialization function for `mcSpec` class is incomplete, in the sense that it does not cover all cases.

Author(s)

Georgi N. Boshnakov

See Also

mcSpec

Examples

```
mcSpec(dim = 5, mo = 4, root1 = c(1,1), order = rep(3,4))
mcSpec(dim = 5, mo = 4, root1 = c(1,1,1), order = rep(5,4))
mcSpec(dim = 5, mo = 4, root1 = c(1,1,1,1), order = rep(5,4))
```

---

mcStable *Check if an object is stable*

Description

Check if an object is stable.

Usage

```
mcStable(x)
```

Arguments

x	the object to be checked

Details

A stable matrix is a matrix all of whose eigenvalues have moduli less than one. Other objects are stable if the associated matrix is stable.

This is a generic function. The default method works as follows. x is a square matrix, the method checks if its eigenvalues satisfy the stability condition and returns the result.

Otherwise, if x is a rectangular matrix with more columns than rows, it is assumed to be the top of a multi-companion matrix. If x is a vector, it is assumed to represent the top row of a companion matrix. In all other cases x is converted to matrix with `as.matrix(x)`. The result should be a square matrix whose eigenvalues are checked. It is an error for the matrix to have more rows than columns.

Value

TRUE if the object is stable and FALSE otherwise

Note

An argument . . . may be a good idea since methods may wish to provide options. For example, for continuous time systems, the stability condition is that the real parts of the eigenvalues are negative.

For example, an option to choose the left half-plane for the stable region, instead of the unit circle, would handle stability for continuous time systems.

Author(s)

Georgi N. Boshnakov

Examples

```
## a simulated matrix (it is stable by default)
mc <- mCompanion("sim", dim=4, mo=2)
mcStable(mc)

## a square matrix
m <- matrix(1:9, nrow=3)
eigen(m)$values
mcStable(m)

## a 2x4 matrix, taken to be the top of an mc matrix
m <- matrix(1:8, nrow=2)
mcStable(m)
mCompanion(m)

## a vector, taken to be the top row of an mc matrix
v <- 1:4
mcStable(v)
mCompanion(v)
abs(mc_eigen(mCompanion(v))$values)

co1 <- cbind(c(1,1,1,1), c(0,1,0,0))

## a matrix with eigenvalues equal to 1
mat2 <- make_mcmatrix(eigval = c(1), co = co1, dim = 4, len.block = c(2))
## mat2 is ordinary matrix, eigenvalues are computed numerically
eigen(mat2)
mcStable(mat2)  # FALSE but in general depends on floating point arithmetic

mat2a <- mCompanion(x="gen", eigval = c(1), co = co1, dim = 4, len.block = c(2), what.res = "list")
mc_eigen(mat2a)
mcStable(mat2a)


mat2b0 <- make_mcmatrix(eigval = c(1), co = co1, dim = 4, len.block = c(2), what = "list")
mat2b <- mCompanion(mat2b0, "gen")
mc_eigen(mat2b)
mcStable(mat2b)

## mat2c is a MultiCompanion object with the eigenvalues stored in it
```

```
mat2c <- mCompanion(x="sim", eigval = c(1,0,0), co = cbind(co1, c(0,0,1,0), c(0,0,0,1)),
                    dim = 4, len.block = c(2,1,1))
mat2c
## since the eigenvalues are directly available here, no need to compute them
mc_eigen(mat2c) # contains a 2x2 Jordan block.
mcStable(mat2c)
```

`mc_chain_extend` *Extend multi-companion eigenvectors*

Description

Extend Jordan chains of a multi-companion matrix to higher dimension and complete them to a full system by adding eigenchains for zero eigenvalues.

Usage

```
mc_chain_extend(ev, newdim)
```

Arguments

`ev`	eigenvalues and eigenvectors, a list with components `values` and `vectors`.
`newdim`	the new dimension of the vectors.

Details

The eigenvectors of a multi-companion matrix have a special structure. This function extends the supplied eigenvectors to be eigenvectors of a higher-dimensional multi-companion matrix of the same multi-companion order with the same top rows extended with zeroes.

`ev` is a list with components `values`, `vectors` and possibly others. In particular, `ev` may be the value returned by a call to the base function `eigen()`. A component `len.block` may be used to specify the lengths of the Jordan chains, by default all are of length one.

The function handles also the case when only the first `mo.col` columns of the top of the original multi-companion matrix are non-zero. This may be specified by a component `mo.col` in `ev`, otherwise `mo.col` is set to the dimension of the space spanned by the non-zero eigenvalues.

When `mo.col` is smaller than the multi-companion order, the information in the eigenvectors is not sufficient to extend them. The missing entries are supplied via the argument `F0bot` (**TODO: describe!**).

Chains corresponding to zero eigenvalues come last in the result.

Value

The eigenvectors extended to the new dimension.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

mc_eigen;

the main work is done by mC.non0chain.extend and mc_0chains.

Examples

```
ev <- make_mcchains(eigval = c(1, 0.5), co = cbind(c(1,1), c(1, -1)), dim = 4,
              mo.col = 2,
              len.block = c(1, 1))
ev
## extend evecs in ev to the requested dim and complete with chains for eval 0.
mc_chain_extend(ev = ev, newdim = 6)
mc_chain_extend(ev = ev, newdim = 7)
```

---

mc_eigen *The eigen decomposition of a multi-companion matrix*

Description

Give the eigenvalues or the entire eigen decomposition of a multi-companion matrix

Usage

```
mc_eigen(x, ...)
mc_eigenvalues(x, ...)
```

Arguments

x	a multi-companion matrix, an object of class MultiCompanion.
...	additional arguments, currently not used.

Details

Both functions first check if the decomposition is stored in x and, if that is the case, return the result without computations. This is particularly useful when the matrix is created from its spectral decomposition in the first place. The only restrictions on the result in this case come from the structure of multi-companion matrices.

Otherwise they use eigen to do the main computation. In addition, if the top of the matrix has structural columns of zeroes, mc_eigen takes care to call eigen with a sub-matrix whose last column is not zero, and handles the zero eigenvalues separately.

Note that x@mo.col is the last column containing nonzero elements in the top of the matrix. By calling eigen on the top left x@mo.col square block, rather than on the entire matrix, we achieve several things. Firstly, this block may turn out to be non-singular. In that case, the chains corresponding to zero eigenvalues, if any, are structural and straightforward. Secondly, if this block turns out to be singular, we know that by reducing the dimension we have left out only elements corresponding to zero eigenvalues. The vectors associated with zero eigenvalues are somewhat tricky in this case, but manageable.

The net effect is that the only restriction comes from the use of eigen, which does not handle Jordan chains of length larger than one. In general, this is not a problem, since chains with more than one vector are not likely to occur numerically. In particular, it is relatively safe to assume that the space spanned by the non-zero eigenvalues of the multicompanion matrix has a basis of eigenvectors. However, when x@mo.col is smaller than the dimension of the matrix, eigenchains associated with the zero value can easily occur, due to the structure of the matrix. That is why we pay special attention to them.

In mc_eigen the handling of the zero eigenvalues is based on mc_chain_extend. The latter takes care also of zero eigenvalues whose Jordan blocks are of size larger than one.

Value

For mc_eigenvalues, the eigenvalues as a vector.

For mc_eigen, the eigenvalues and eigenvectors as a list with components values and vectors. In addition the list contains a component len.block with the lengths of the Jordan chains.

Note

mc_eigenvalues currently simply calls eigen if the eigenvalues are not stored in the object. It is probably mostly useful when the interest is in the nonzero eigenvalues.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Examples

```
x <- sim_mc(6,4,mo.col=2)
x
y <- mCompanion(x,detect="gen")
y
z <- as.matrix(y)
xx <- mCompanion(x=z,mo.col=2)
mc_eigen(xx)
```

`mc_factorize` *Factorise multi-companion matrices*

Description

Companion factorization of multi-companion matrices.

Usage

```
mc_factorize(x, mo, mo.col)
mc_leftc(x, mo, mo.col)
```

Arguments

`x`	a multi-companion matrix or its top.
`mo`	multi-companion order, number of structural top rows.
`mo.col`	number of non-trivial columns in the top of the matrix.

Details

The companion factorization of a multi-companion matrix, X, of (multi-companion) order p is $X = A_1 \times \cdots \times A_p$, where A_i, $i = 1, \ldots, p$, are companion matrices.

`mc_leftc` factorises a multi-companion matrix into a product of companion times multi-companion.

`mc_factorize` calls `mc_leftc` a number of times to compute the full factorisation.

If `x` is not a matrix an attempt is made to convert it to matrix. If `x` is a vector it is converted to a matrix with 1 row.

`x` may be the whole matrix or its top. If `mo` is missing `x` is assumed to be the top of the matrix and the multi-companion order is set to its number of rows.

`mo.col` defaults to the number of columns of `x`. It is important to specify `mo.col` if there are columns of zeroes in the top of the matrix. Otherwise the factorisation usually fails with a message (from `solve`) that the system is exactly singular. Note however that for objects of class MultiCompanion this situation is handled automatically (unless the user overwrites the default behaviour).

Value

for `mc_factorize`, a matrix whose i-th row is the first row of the i-th companion factor.

for `mc_leftc`, a numeric vector containing the first row of the companion factor.

Level

0

Note

The companion factorisation does not always exist but currently this possibility is not handled. Even if it exists, it may be numerically unstable.

Also, if `mo.col` is smaller than the number of columns, then the factorisation is not unique, the one having `mo.col` non-zero entries is computed. The existence is not treated.

`mc_leftc` is probably the first function I wrote for multi-companion matrices. It does not do checks consistently. The MultiCompanion class can be used here.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

See Also

`mc_from_factors`

Examples

```
mat2 <- make_mcmatrix(eigval = c(1), co = cbind(c(1,1,1,1), c(0,1,0,0)), dim = 4, len.block = c(2))
mat2
eigen(mat2)
mc_leftc(mat2, mo = 4, mo.col = 2)
mCompanion(mat2)
mCompanion(mat2, mo=4, mo.col=2)
mc_leftc(mCompanion(mat2), mo = 4, mo.col = 2)
mc_eigen(mCompanion(mat2), mo = 4, mo.col = 2)
mc_eigen(mCompanion(mat2, mo=4, mo.col=2), mo = 4, mo.col = 2)
```

---

mc_factors *Factors of multi-companion matrices*

Description

Gives the factors comprising the companion factorisation of a multi-companion matrix.

Usage

```
mc_factors(x, what = "mc")
```

Arguments

`x`	a multi-companion matrix, an object of class `MultiCompanion`.
`what`	format of the result, see below.

Details

If the factors are available in the object's pad in the requested format, they are returned without further processing. The factors may be available if they have been previously computed or if the matrix has been created from the factors.

If the factors are available, but not in the requested format, they are converted to it. Otherwise the factors are computed.

The factors are stored in the object's pad under the name "mC.factors" when `what == "mc"`, and in "mC.factorsmat" otherwise.

Value

If `what == "mc"` the companion factors of `x` as a list of `MultiCompanion` objects.

Otherwise a matrix with i-th row representing the i-th factor.

As a side effect, the factors are stored in the object's pad, see 'Details'.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Examples

```
m <- mCompanion(matrix(1:8, nrow = 2))
mc_factors(m)
```

`mc_from_factors` *Multi-companion matrix from factors*

Description

Compute a multi-companion matrix from its companion factors or from a periodic filter. Create the multi-companion matrix corresponding to a periodic filter by multiplying the relevant companion matrices in reverse order.

Usage

```
mc_from_factors(x)
mc_from_filter(x)
```

Arguments

`x`	a matrix with a row for each companion factor, see details.

Details

x is a matrix whose i-th row is the top row of the i-th companion factor (for mc_from_factors) or the filter coefficients for the i-th season (for mc_from_filter).

mc_from_factors is, effectively, the inverse of mc_factorize. The companion matrices specified by the argument are multiplied.

mc_from_filter is similar except that the relevant companion matrices are multiplied in reverse order. After all, it is natural to have the coefficients for the i-th season in the i-th row!

todo: add an argument to specify the "first" season.

Value

The top of the resulting multi-companion matrix.

Level

Currently mc_from_factors calls mCompanion, which it probably should not do.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

See Also

mc_factorize

Examples

```
x <- matrix(1:8, nrow = 2)
mc_from_factors(x)
mCompanion(mc_from_factors(x))

mc_from_filter(x)
mCompanion(mc_from_filter(x))
```

mc_matrix *Basic utilities for multi-companion matrices*

Description

Compute the dense matrix representation of a multi-companion matrix or convert the argument to an ordinary matrix.

Usage

```
mc_full(x)
mc_matrix(x)
mc_order(x)
is_mc_bottom(x)
```

Arguments

x	the top part of the multi-companion matrix or the whole matrix, see Details.

Details

`mc_matrix` returns an ordinary matrix. It returns `x` if `x` is an ordinary matrix (`is.matrix(x) == TRUE`), converts `x` to a matrix with one row if `x` is a vector, and returns `as.matrix(x)` otherwise. `mc_matrix` is used by some functions in package `mcompanion` that want to allow flexible format for the top of a multicompanion matrix or even the whole matrix (e.g. `x` may be a `MultiCompanion` object) but are not really multi-companion aware.

For `mc_full`, `x` is normally the top part of a multi-companion matrix. Rows are appended as necessary to obtain the dense representation of the matrix and the result is guaranteed to be a multi-companion matrix. It is an error to have more rows than columns. If the number of rows is equal to the number of columns, i.e. `x` is the whole matrix, the effect is that `x` is converted to an ordinary matrix but no check is made to see if the result is indeed a multi-companion matrix. `x` may be a vector if the multi-companion order is 1.

Give the multi-companion order of a square matrix

Determine the multi-companion order of a square matrix or check if a matrix may be the bottom part of a multi-companion matrix.

In `mc_order(x)` should be a square matrix, while in `is_mc_bottom(x)` the matrix is usually rectangular.

The bottom part of a multi-companion matrix is of the form [I 0], where I is an identity matrix and 0 is a matrix of zeroes. The top consists of the rows above the bottom part. The multi-companion order is the number of rows in the top of a multi-companion matrix.

Identity matrices have `mc_order` zero. Other general matrices have `mc_order` equal to the number of rows. In particular, an 1×1 matrix has `mc_order` zero, if its only element is equal to one, and `mc_order` one otherwise.

Acordingly, `is_mc_bottom(x)` returns TRUE if `x` is the identity matrix or a matrix with zero rows. This is consistent with the treatment of the identity matrix as multi-companion of multi order 0 and a general matrix as multi-companion of multi-companion order equal to the number of its rows.

Value

for `mc_full`, the multi-companion matrix as an ordinary dense matrix object.

For `mc_matrix`, an ordinary matrix.

for `mc_order`, the multi-companion order of `x`, a non-negative integer

for `is_mc_bottom`, TRUE if `x` may be the bottom part of a multi-companion matrix and FALSE otherwise.

Note

`mc_matrix` is not multi-companion specific, except that it converts a vector to a matrix with one row (not column). For square matrices these functions are not really multi-companion specific.

It may make sense to allow non-square matrices also for `mc_order`.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

See Also

`mcStable`

Examples

```
mc <- mCompanion("sim", dim = 4, mo = 2)
mc
mc_order(mc)
x <- mc[1:2, ]   # the top of mc
x
x2 <- mc[]      # whole mc as ordinary matrix
x2

mc_matrix(mc)
mc_matrix(x2)
## mc_matrix() doesn't append rows to its argument
mc_matrix(x)

## mc_full() appends rows, to make the matrix square multicompanion
mc_full(x)
## mc and x2 are square, so not amended:
mc_full(mc)
mc_full(x2)

## a vector argument is treated as a matrix with 1 row:
mc_matrix(1:4)
mc_full(1:4)
```

```
## mc_order(1:4) # not by mc_order


m <- mCompanion(matrix(1:8, nrow = 2))
mc_matrix(m)
mc_order(m)

m[-c(1,2), ]
is_mc_bottom(m[-c(1,2), ]) # TRUE

## TRUE for reactangular diagonal matrix with nrow < ncol
is_mc_bottom(diag(1, nrow = 3, ncol = 5))
## border cases
is_mc_bottom(matrix(0, nrow = 0, ncol = 4)) # TRUE, 0 rows
is_mc_bottom(diag(4))                       # TRUE, square diagonal matrix
```

`mf_VSform` *Extract properties of multi-filters*

Description

Extract properties for scalar and vector of seasons forms of multi-filters.

Usage

```
mf_order(x, i = "max", form = "pc", perm)
mf_period(x)
mf_poles(x, blocks = FALSE)
mf_VSform(x, first = 1, form = "U", perm)
```

Arguments

`x`	the filter, an object of class `"MultiFilter"`.
`i`	index, integer vector or a string.
`first`	the first season of the year.
`form`	the form of the filter to which the result refers, one of "pc", "I", "U", or "L", see Details.
`perm`	permutation of the seasons within the year.
`blocks`	request lengths of Jordan chains.

Details

With the default `i=="max"` the function `mf_order` returns a single number, the order of the filter in the representation requested by `form`. The orders of the components may be obtained with the setting `i=="all"` which gives a vector whose j-th element is the order of the j-th component of the filter. A subset of these may be obtained with numeric `i` which is treated as standard index vector. Values for `i` other than the default are meaningful mainly for `form="pc"`.

mf_VSform arranges the filter coefficients in one of the vector of seasons forms (todo: cite me). The component Phi of the result is a matrix obtained by putting the coefficient matrices next to each other, [A1 ... Ad]. If perm is provided, then the result is the same for "U" and "L".

mf_VSform is called implicitly by the subscripting operation ("[") when needed, it is more flexible and is recommended for general use.

For the vector forms ("I", "U", and "L") the argument perm specifies the arrangement of the components of the filter in that form. For the I- and U-forms the default is mf_period(x):1, for the L-form it is 1:mf_period(x).

Currently perm may take on values that can be obtained from the default by rotation, e.g. if the period is 4, perm may be one of (4,3,2,1), (1,4,3,2), (2,1,4,3), (3,2,1,4) for the U-form, and (1,2,3,4), (4,1,2,3), (3,4,1,2), (2,3,4,1) for the L-form. Other permutations may be usefull in some situations but may not result in U- or L- forms (without further transformations). For I-form any permutation should be permissible when implemented (todo:).

For mf_order the argument perm affects the computation only, not the ordering in the result. The result (if vector) is not permuted unless the argument i asks for this. For mf_VSform however such a behaviour would be very peculiar and the rows of the result are for the permuted seasons. In short, the i-th element of the result of mf_order (if vector) gives the order (in the requested form) of the i-th season but the i-th row of any of the matrices returned by mf_VSform depends on perm and form.

Note: the terminology here reflects application to pc processes, probably should be made more neutral in this respect.

todo: (2013-03-26) mf_order seems unfinished.

Value

For mf_order, if i = "max" a positive integer, otherwise a vector of positive integers.

For mf_period the period of the filter, a positive integer.

For mf_poles, if blocks = FALSE, a vector of the eigenvalues of the associated multi-companion matrix, each eigenvalue repeated according to its algebraic multiplicity. If blocks = TRUE, a 2-column matrix with the eigenvalues in the first column and the lengths of the Jordan chains in the second. There is one row for each chain (i.e. multiple eigenvalues are repeated according to their geometric multiplicity).

For mf_VSform a list with components:

Phi0	the zero lag coefficient, a matrix,
Phi	the remaining coefficients, a matrix,
Phi0inv	(form=="I" only) the inverse of the zero lag coefficient matrix of the vs-form, a matrix. (**TODO:** the name of this component is misleading since in the case form = "I" Phi0 is the identity matrix and Phi0inv is not equal to the inverse of Phi0.)

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

MultiFilter and the examples there, mcStable

Examples

```
## simulate a 3x3 2-companion matrix
##  and turn it into a multi-filter
(m <- mCompanion("sim", dim=3, mo=2))
(flt <- new("MultiFilter", mc = m ))
mf_period(flt)
mf_poles(flt)
abs(mf_poles(flt))
mf_VSform(flt,form="U")
mf_VSform(flt,form="L")
mf_VSform(flt,form="I")

## simulate a pc filter (2 seasons)
## and turn it into a multi-filter object
(rfi <- sim_pcfilter(2, 3))
(flt <- new("MultiFilter", coef = rfi$pcfilter))
mf_period(flt)
mf_poles(flt)
abs(mf_poles(flt))
mf_VSform(flt, form="U")
mf_VSform(flt, form="I")
mf_VSform(flt, form="L")

## indexing can be used  to extract filter coefficients
flt[]
flt[1,]
## the rest are some checks of numerical performance.
rfi
rfi$mat==0

zapsmall(rfi$mat)
mCompanion(zapsmall(rfi$mat))
unclass(mCompanion(zapsmall(rfi$mat)))
unclass(mCompanion(rfi$mat))

flt1 <- new("MultiFilter", mc = mCompanion(zapsmall(rfi$mat)))
flt2 <- flt

flt1[]
flt2[]
```

```
flt1[] - flt2[]
rfi$pcfilter - rfi$mat[1:2,]

mf_poles(flt1)
abs(mf_poles(flt1))

svd(rfi$mat)
rcond(rfi$mat)
Matrix::rcond(Matrix::Matrix(rfi$mat),"O")
1/Matrix::rcond(Matrix::Matrix(rfi$mat),"O")
```

`MultiCompanion-class` *Class "MultiCompanion"*

Description

Objects and methods for multi-companion matrices

Objects from the Class

For ordinary usage objects from this class should behave as matrices and there should be no need to access the slots directly.

Objects can be created with the function `mCompanion`. Other functions in the mcompanion package also produce `MultiCompanion` objects.

It is possible also to call `new()` directly:

```
new("MultiCompanion", xtop, mo, n, mo.col, ido, x, dimnames,
    detect, misc)
```

Arguments:

`xtop` is the top of the matrix.

`mo` is the multi-companion order of the matrix.

`n` is the dimension.

`mo.col` is the top order, meaniing that columns mo.col+1,...,n of the top of the matrix are zeros. `mo.col` may also be set to "detect", in which case it is determined by scanning `xtop` or `x`.

`ido` the dimension of the identity sub-matrix.

`x` the whole matrix.

`dimnames` is not used currently.

`detect` controls whether automatic detection of `mo` and `mo.col` should be attempted. The values tested are "mo", "mo.col", "all", and "nothing" with obvious meanings.

`misc` todo: describe this argument!

Normally one of xtop and x is supplied but if both are, they are checked for consistency, including the elements of the matrix (equality is tested with ==). To facilitate calls with one unnamed argument, when xtop is a square matrix it is taken to be the entire matrix (provided that x is missing).

Aside from xtop (or x), most of the remaining arguments can be deduced automatically. The number of rows and columns of xtop give the multi-companion order and the dimension of the matrix, respectively. A vector xtop is taken to stand for a matrix with one row. x needs to be square or a vector of length equal to exact square. mo and mo.col may be determined from the contents of x and xtop. There is no harm in ignoring mo.col but it is useful for our applications. Note that by default it is to set to the number of columns and not determined by scanning the matrix.

The contents of the misc argument are stored in the pad of the new object.

Slots

xtop: The top of the matrix, an object of class "matrix"

mo: Multi-companion order, an object of class "numeric"

ido: dimension of the identity submatrix, object of class "numeric"

mo.col: number of non-zero columns in top rows, object of class "numeric"

pad: storage for additional info, object of class "objectPad"

x: inherited, object of class "numeric"

Dim: inherited, object of class "integer"

Dimnames: inherited, object of class "list"

factors: inherited, object of class "list"

Extends

Class "ddenseMatrix", directly. Class "generalMatrix", directly. Class "dMatrix", by class "ddenseMatrix". Class "denseMatrix", by class "ddenseMatrix". Class "Matrix", by class "ddenseMatrix". Class "Matrix", by class "ddenseMatrix". Class "compMatrix", by class "generalMatrix". Class "Matrix", by class "generalMatrix".

Methods

%*% signature(x = "ANY", y = "MultiCompanion"): ...

%*% signature(x = "MultiCompanion", y = "MultiCompanion"): ...

%*% signature(x = "MultiCompanion", y = "ANY"): ...

[signature(x = "MultiCompanion", i = "index", j = "index", drop = "logical"): ...

[signature(x = "MultiCompanion", i = "index", j = "missing", drop = "logical"): ...

[signature(x = "MultiCompanion", i = "missing", j = "index", drop = "logical"): ...

coerce signature(from = "dgeMatrix", to = "MultiCompanion"): ...

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

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

coerce signature(from = "MultiCompanion", to = "dgeMatrix"): ...

initialize `signature(.Object = "MultiCompanion")`: This method is called implicitly when the user calls `new("MultiCompanion",...)`.

mcStable `signature(x = "MultiCompanion")`: ...

t `signature(x = "MultiCompanion")`: ...

%*% `signature(x = "matrix", y = "MultiCompanion")`: ...

%*% `signature(x = "MultiCompanion", y = "matrix")`: ...

[`signature(x = "MultiCompanion", i = "index", j = "index", drop = "missing")`: ...

[`signature(x = "MultiCompanion", i = "index", j = "missing", drop = "missing")`: ...

[`signature(x = "MultiCompanion", i = "missing", j = "index", drop = "missing")`: ...

%*% `signature(x = "MultiCompanion", y = "vector")`: ...

%*% `signature(x = "vector", y = "MultiCompanion")`: ...

Note

The implementation is rather redundant, this class probably should inherit in a different way from classes in Matrix package or may be not inherit at all.

Methods to get the multi-order, mo.col, and others, would be useful but first the terminology needs to be made consistent.

Other matrix arithmetic operations?

Argument `n` is called `dim` in other functions.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

`mCompanion` and the examples there; the following functions produce multi-companion matrices but do not return `MultiCompanion` objects:

`sim_mc`, `make_mcmatrix`

Examples

```
a1 <- matrix(1:12,nrow=2)
mc1 <- new("MultiCompanion",xtop=a1)
new("MultiCompanion",a1)   # same

a2 <- matrix(c(1:6,rep(0,4)),nrow=2)   # 1st 3 columns of a2 are non-zero
mc2 <- new("MultiCompanion",a2)
```

```
mc2
mc2@mo.col     # =5, because the default is to set mo.col to ncol

mc2a <- new("MultiCompanion",a2,detect="mo.col")
mc2a@mo.col   # =3, compare with above

b <- as(mc2,"matrix")  # b is ordinary R matrix
mcb <- new("MultiCompanion",x=b)
       new("MultiCompanion",b)   # same as mcb

mcb@mo        # 2 (mo detected)
mcb@mo.col    # 5 (no attempt to detect mo.col)

mcba <- new("MultiCompanion",b,detect="all")
mcba@mo        # 2 (mo detected)
mcba@mo.col    # 3 (mo.col detected)
```

`MultiFilter-class` *Class "MultiFilter"*

Description

Objects and methods for filters with more than one set of coefficients.

Objects from the Class

Objects can be created by calls of the form `new("MultiFilter", coef, mc, order, sign)`.

Objects from this class represent periodic filters. A d-periodic filter relates an input series ε_t to an output series y_t by the following formula:

$$y_t = \sum_{i=1}^{p_t} \phi_t(i) y_{t-i} + \varepsilon_t,$$

where the coefficients $\phi_t(i)$ are d-periodic in t, i.e. $\phi_{t+d}(i) = \phi_t(i)$ and $p_{t+d} = p_t$.

The periodicity means that it is sufficient to store the coefficients in a $d \times p$ matrix, where $p = \max(p_1, \ldots, p_t)$. Slot `coef` contains such a matrix.

The filter may be specified either by its coefficients or by its multi-companion form.

Slots

`mc`: the multi-companion form of the filter, an object of class `"MultiCompanion"`

`coef`: the coefficients of the filter, an object of class `"matrix"`, whose sth row contains the coefficients for $t = k \times d + s$.

`order`: the periodic order of the filter, a numeric vector giving the orders of the individual seasons.

`sign`: 1 or -1. The default value, 1, corresponds to the formula given in section "Objects from the Class". It can also be -1, if the sum on the right-hand side of that formula is preceded by a minus (usual convention in signal processing).

Methods

[`signature(x = "MultiFilter", i = "ANY", j = "ANY",drop = "ANY")`: take subset of the coefficients of the filter in various forms.

To do: the function needs more work! Document the function and the additional arguments!

initialize `signature(.Object = "MultiFilter")`: This function is called implicitly by `new`, see the signature for `new` above. One of `mc` and `coef` must be supplied, the other arguments are optional.

If `mc` is missing it is computed from `coef`. In this case, component `mC.factorsmat` of slot `misc` of `mc` is set to the companion factorisation of `mc` (essentially the reversed rows of `coef`).

If `coef` is missing it is computed from `mc`, see `mc_factors`.

mcStable `signature(x = "MultiFilter")`: Check if the filter is stable.

See also the documentation for the following functions which are effectively methods for class "MuliFilter" but are not defined as formal methods:

`mf_period`, `mf_order`, `mf_poles`, `mf_VSform`.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

`MultiCompanion`, `mf_period`, `mf_order`, `mf_poles`, `mf_VSform`.

Examples

```
showClass("MultiFilter")

m <- mCompanion("sim",dim=3,mo=2)   # simulate a 3x3 2-comp. matrix
flt <- new("MultiFilter", mc = m )
flt[]
mf_period(flt)
mf_poles(flt)
abs(mf_poles(flt))

mf_VSform(flt,form="U")
mf_VSform(flt,form="L")
mf_VSform(flt,form="I")


# try arguments "coef" and "mc", for comparison
rfi <- sim_pcfilter(2,3) # period=2, order=c(3,3)
# per. filter from a multi-companion matrix
```

```
flt1 <- new("MultiFilter",mc=  mCompanion(zapsmall(rfi$mat)) )
flt1[]
mf_period(flt1)
mf_poles(flt1)
abs(mf_poles(flt1))

mf_VSform(flt1,form="U")
mf_VSform(flt1,form="L")
mf_VSform(flt1,form="I")

# per. filter from coefficients, should be the same (numerically)
flt2 <- new("MultiFilter",coef=rfi$pcfilter)
flt2[]
mf_period(flt2)
mf_poles(flt2)
abs(mf_poles(flt2))

mf_VSform(flt2,form="U")
mf_VSform(flt2,form="L")
mf_VSform(flt2,form="I")
```

`null_complement` *Compute the orthogonal complement of a subspace*

Description

Computes the orthogonal complement of a subspace relative to a universe.

Usage

```
null_complement(m, universe = NULL, na.allow = TRUE)
```

Arguments

`m`	NA or a matrix whose columns define the subspace, a vector is treated as a matrix with one column.
`universe`	a matrix whose columns specify the subspace relative to which to compute the complement, the default is the full space.
`na.allow`	if `TRUE`, default, treat NA's specially, see Details.

Details

`null_complement` computes the orthogonal complement of a subspace (spanned by the columns of `m`) relative to a universe.

Argument `universe` can be used to specify a subspace w.r.t. which to compute the complement. If `universe` is `NULL` (the default), the complement w.r.t. the full space is computed. The full space is the n-dimensional space, where n is the number of rows of argument `m`.

`null_complement` returns a matrix whose columns give a basis of the required subspace.

`null_complement` uses `Null()` from package MASS for the actual computation. `null_complement(m, na.allow = FALSE)` is equivalent to `Null(m)`.

`m` is typically a matrix whose columns represent the subspace w.r.t. which to compute the complement. `null_complement` can also deal with NA's in `m`. This facility can be turned off by specifying `na.allow = FALSE`.

If `na.allow = TRUE`, the default, and `m` is identical to `NA`, `universe` is returned (i.e. `m = NA` represents the empty subspace). Note that in this case `universe` cannot be NULL, since there is no way to determine the dimension of the full space.

Otherwise, `m` is a matrix. If all elements of `m` are `NA`, a matrix of NA's is returned with number of columns equal to `ncol(universe) - ncol(m)`.

Value

a matrix representing a basis of the requested subspace

Author(s)

Georgi N. Boshnakov

Examples

```
m1 <- diag(1, nrow = 3, ncol = 2)
null_complement(m1)

null_complement(c(1,1,0))
null_complement(c(1,1,0), m1)

## the columns of the result from null_complement() are orthogonal
## to  the 1st argument:
t(c(1,1,0)) %*% null_complement(c(1,1,0))
t(c(1,1,0)) %*% null_complement(c(1,1,0), m1)

null_complement(rep(NA_real_, 3), m1)
null_complement(NA, m1)
```

---

permute_var *Permute rows and columns of matrices*

Description

Permute rows and columns of matrices.

Usage

```
permute_var(mat, perm = nrow(mat):1)
permute_synch(param, perm)
```

Arguments

mat	a matrix.
param	a matrix or list, see Details.
perm	permutation, defaults to nrow:1.

Details

Given a permutation, permute_var permutes the rows and columns of a matrix in such a way that if mat is the covariance matrix of a vector x, then the rearranged matrix is the covariance matrix of x[perm]. If P is the permutation matrix corresponding to perm, then the computed value is P %*% mat %*% t(P).

permute_synch performs the above transformation on all matrices found in param. More precisely, if param is a matrix, then the result is the same as for permute_var. Otherwise param should be a list and, conceptually, permute_synch is applied recursively on each element of this list. The net result is that each matrix, say M, in param is replaced by PMP' and each vector, say v, by Pv. The idea is that param may contain specification of a VAR model, all components of which need to be reshuffled if the components of the multivariate vector are permuted.

All matrices in param must have the same number of rows, say d, but this is not checked. perm should be a permutation of 1:d.

Value

for permute_var, a matrix,

for permute_synch, a matrix or list of the same shape as param in which each matrix is transformed as described in Details.

Author(s)

Georgi N. Boshnakov

Examples

```
Cl <- cor(longley) # from example for 'cor()'
nc <- ncol(Cl)
v <- 1:nc
names(v) <- colnames(Cl)

permute_var(Cl)
all(permute_var(Cl) == Cl[ncol(Cl):1, ncol(Cl):1])
```

rblockmult *Right-multiply a matrix by a block*

Description

Treats a matrix as a block matrix and multiplies each block by a given block.

Usage

```
rblockmult(x, b)
```

Arguments

x	the matrix.
b	the block.

Details

x is split into blocks [x1 ... xn] so that ncol(xi)==nrow(b) and each block is multiplied by b. The result is the matrix [x1 b ... xn b].

Value

the matrix obtained as described above

Author(s)

Georgi N. Boshnakov

Examples

```
m <- matrix(1:12, nrow = 2)
b <- matrix(c(0, 1, 1, 0), nrow = 2)
rblockmult(m,b)
```

sim_mc *Simulate a multi-companion matrix*

Description

Simulate a multi-companion matrix with partially or fully specified spectral properties.

Usage

```
sim_mc(dim, mo, mo.col = dim, eigval, len.block, type.eigval = NULL,
       co, eigabs, eigsign, type = "real",
       value = "real", value.type = "", ...)
```

Arguments

dim	dimension of the matrix.
mo	multi-companion order.
mo.col	number of structural columns.
eigval	eigenvalues, one for each Jordan block.
len.block	lengths of the Jordan blocks corresponding to eigval.
type.eigval	types of the eigenvalues, a character vector
co	co parameters, see Details.
eigabs	moduli (absolute values) of eigenvalues, see Details.
eigsign	signs or complex arguments of eigenvalues, see Details.
type	passed down to generators (???)
value	what to return
value.type	type of the value (???)
...	further arguments to passed on to sim_chains and sim_numbers, see Details.

Details

sim_mc generates a multi-companion matrix of dimension dim x dim and multi-companion order mo. The matrix has the spectral properties specified by the arguments. Values that cannot be inferred from the arguments are simulated.

Arguments dim, mo, and mo.col define the structure of the matrix. The first two are compulsory but the last one, mo.col, is optional. If no other arguments are supplied sim_mc produces a matrix with all spectral parameters simulated.

The number of non-zero eigenvalues is at most mo.col. If mo.col < dim the multi-companion matrix has structural eigenvectors/chains corresponding to the zero eigenvalue(s), see the references. These chains are generated automatically.

Arguments type.eigval, eigabs, eigsign and eigval are vectors used to specify the types and the values of the eigenvalues. Any or all of them may be missing or NULL. Those present must have the same length.

It is not necessary to specify eigenvalues and eigenvectors corresponding to eigenvalues equal to zero, since the structural eigenchains needed when mo.col < dim are created automatically. In practice, the number of the non-zero eigenvalues is usually equal to mo.col. The net effect is that the arguments specifying the spectral structure of the matrix normally need to specify the spectral information about the non-zero eigval only.

Some or all of the eigenvalues may be specified partially or fully using arguments eigabs, eigsign, and eigval. Non-NA entries in eigval specify complete eigenvalues. Non-NA entries in eigabs specify absolute values of eigenvalues. Non-NA entries in eigsign specify signs of real eigenvalues or complex arguments of complex eigenvalues. Generally, if the entry for an eigenvalue in eigval is a number (not NA), then the corresponding entries in eigabs and eigsign will be NA. This is not enforced and a limited check for consistency is made in case of redundant information.

type.eigval is a character vector describing the types of the eigenvalues, where "r", "c", and "cp" stand for real, complex, and complex pair, respectively. It is best to have one entry only for each complex pair (specified by "cp"), rather than two "c" entries.

If `type.eigval` is NULL (default) and `eigval` is supplied, then `type.eigval` is inferred from the imaginary part of `eigval` ("r" or "cp"), if it is complex.

For compatibility with older versions of this function `eigval` may be a character vector in which case it is simply assigned to `type.eigval`.

If both, `type.eigval` and `eigval`, are missing a default allocation of the types of the eigenvalues is chosen.

TODO: complete the description below.

The remaining spectral parameters may be specified with the argument `co` with missing entries for the "free" entries. (!!! This is not complete, it may be better to have separate arguments for the absolute value and the angle, as for eigenvalues, and an option for normalisation of these coefficients. ???)

Generators other than the default ones may be specified in the `...` argument. These are passed to `sim_numbers` and `sim_chains`. Again, for the "co" arguments the support is not finished.

Value

if `value.type` is the character string `"matrix"`, the required multi-companion matrix. Otherwise, if `value.type=="list"`, a list containing also the spectral information (this list is the same as the one from `make_mcmatrix`)).

Note

A canonical form is needed, especially when there are repeated eigenvalues whose eigenvectors may be chosen to be orthogonal, at least. (nyakade v zapiskite mi tryabva da ima kanonichna forma!)

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

`gbutils::sim_numbers` and `sim_chains` for arguments that control the distributions of the random numbers.

`make_mcmatrix` creates the matrix.

Examples

```
m0 <- sim_mc(3,2)   # simulate 3x3 2-companion matrix
abs(m0$eigval)      # eigval random, so their abs values

# now fix moduli of eigenvalues, and
```

```
# ask for one real ev and one complex pair of ev's
m1 <- sim_mc(3,2,eigabs=c(0.25,0.5), type.eigval=c("r","cp"))
m1$eigval
abs(m1$eigval)

# same as above, since type.eigval happens to be the default
# dim is odd, by default first ev is real, rest are complex pairs
m1a <- sim_mc(3,2,eigabs=c(0.25,0.5))
m1a$eigval
abs(m1a$eigval)

# simulate 6x6 4-companion matrix
# with ev's at the seasonal frequencies (1.57 3.141593 -1.57)
# and random moduli. 3 complex pairs of ev's
m2 <- sim_mc(6,4, eigsign = pi*c(1/2,1,-1/2) )
Arg(m2$eigval)
```

`sim_pcfilter` *Generate periodic filters*

Description

Generates periodic filters.

Usage

```
sim_pcfilter(period, n.root, order = n.root, mo.col, ...)
```

Arguments

`period`	the period.
`n.root`	number of non-zero roots (poles).
`order`	order of the filter.
`...`	additional parameters to be passed down to `sim_mc`.
`mo.col`	the last non-zero column in the top of the mc-matrix. The default is `dim`.

Details

Generates periodic filters using the multicompanion approach (Boshnakov and Iqelan 2009).

By default the generated filter is stable and may be used as the autoregressive or moving average part of a periodic autoregressive moving average model. The filter is generated from the specified spectral information by factoring a multi-companion matrix. Any non-specified quantities are generated randomly. Randomly generated eigenvalues correspond to stable filter. The user may specify non-stable roots, unit roots in particular, see `sim_mc`.

Value

A list as obtained from sim_mc with an addtional component for the filter.

pcfilter a matrix with the filter coefficients for the i-th season in the i-th row.

Note

todo: a) Allow different orders for the individual seasons. This is not trivial and maybe not natural for this method. In the singular case it may make sense to implement different strategies for choosing the factorization (when it is not unique) and to choose more carefully the order of the filter to ensure existence of factorization, see my paper.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN (2002). "Multi-companion matrices." *Linear Algebra Appl.*, **354**, 53–83. ISSN 0024-3795, doi:10.1016/S00243795(01)00475X.

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

sim_mc

Examples

```
rfi <- sim_pcfilter(2,3)
rfi
mo <- cbind(c(1,1),rfi$pcfilter)
mo
```

SmallMultiCompanion-class

A class for small multi-companion matrices

Description

A class for internal use for work with small multi-companion matrices.

Objects from the Class

This class is for internal use.

Objects can be created by calls of the form new("SmallMultiCompanion", Mtop, Mbot, jdMtop, MbotXtop).

Slots

`jdMtop`: Object of class `"JordanDecomposition"` ~~

`Mtop`: Object of class `"matrix"` ~~

`Mbot`: Object of class `"matrix"` ~~

`MbotXtop`: Object of class `"matrix"` ~~

Methods

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

initialize `signature(.Object = "SmallMultiCompanion")`: ...

JordanDecomposition `signature(values = "SmallMultiCompanion", vectors = "missing")`: ...

Author(s)

Georgi N. Boshnakov

See Also

`MultiCompanion`

Examples

```
mat2 <- make_mcmatrix(eigval = c(1), co = cbind(c(1,1,1,1), c(0,1,0,0)), dim = 4,
len.block = c(2))
mat2
## Jordan decomp. of mat2[1:2,1:2]:
x2 <- matrix(c(1,1,-1,0), ncol =2)
jd <- matrix(c(1,0,1,1), ncol = 2)
mat2[1:2,1:2] - x2 %*% jd %*% solve(x2)
jdobj <- JordanDecomposition(values = 1, vectors = x2, heights = 2)

m1 <- new("SmallMultiCompanion", mat2[1:2, 1:2], Mbot = mat2[3:4, 1:2], jdMtop = jdobj)
m1a <- new("SmallMultiCompanion", Mbot = mat2[3:4, 1:2], jdMtop = jdobj)
as.matrix(m1) - as.matrix(m1a)  # (approx.) 0's
```

---

spec_core *Parameterise Jordan chains of multi-companion matrices*

Description

Parameterise the Jordan chains corresponding to a given eigenvalue of a multi-companion matrix.

Usage

```
spec_core(mo, evalue, heights, ubasis = NULL, uorth = NULL, evspace = NULL)
```

Arguments

mo	multi-companion order, a positive integer.
evalue	eigenvalue, a real or complex number.
heights	dimensions of Jordan blocks of evalue, a vector of positive integers.
ubasis	basis of the universe, a matrix.
uorth	orthogonal complement of ubasis w.r.t. the full core basis, see Details.
evspace	The space spanned by the eigenvectors, see Details.

Details

spec_core prepares a canonical representation of the parameters of a multi-companion matrix corresponding to an eigenvalue. Roughly speaking, free parameters are represented by NA's in the returned object. For no-repeated eigenvalues the parameterisation consists of the eigenvalue and the seed parameters of the eigenvector. Even then, for uniqueness some convention needs to be adopted.

So, in general the parameterisation is effectively in terms of subspaces.

TODO: Currently this is not documented and is work in progress, there are only some working notes (rakopis: "Some technical details about the parameterisation of mc-matrices").

Value

a list representing the parameterised chains corresponding to the eigenvalue. Currently it contains the following elements:

evalue

heights

co

core.vectors

param.tall

param.hang

generators

Author(s)

Georgi N. Boshnakov

Examples

```
spec_core(4, 1, c(1,1,1,1))

spec_core(4, 1, c(2,1,1,1))
spec_seeds1(c(2,2,2,2), 4)
spec_seeds1(c(2,1,1,1), 4)
spec_core(4, 1, c(2,1,1,1))$co
spec_core(4, 1, c(2,1,1,1))$generators
```

`spec_root0` *Give the spectral parameters for zero eigenvalues of mc-matrices*

Description

Give the spectral parameters for zero eigenvalues of mc-matrices.

Usage

```
spec_root0(dim, mo, mo.col)
```

Arguments

`dim`	dimension of the matrix, a positive integer.
`mo`	multi-companion order, a positive integer.
`mo.col`	last non-zero column in the top of the mc-matrix, a non-negative integer.

Details

`spec_root0` prepares a structure for the zero roots of an mc-matrix.

Value

a list with the following components:

`mo`	multi-companion order
`ev.type`	type of the eigenvalues
`co.type`	not used currently (**:todo:**)
`n.root`	number of non-zero roots
`ev.abs`	absolute values of roots
`ev.arg`	arguments of eigenvalues (0 for positive ev)
`block.length`	lengths of Jordan blocks
`co.abs`	absolute values of seed parameters
`co.arg`	arguments of seed parameters (Hz: 0 for positive; 1/2 for negative)
`co0`	redundant but keep it for now.

Author(s)

Georgi N. Boshnakov

See Also

`spec_root1`, `mcSpec`

Examples

```
spec_root0(4,2,3)
spec_root0(4,2,2)
spec_root0(4,2,1)
spec_root0(5,2,3)
spec_root1(4,2,2)

spec_root0(6,4,2)
spec_root0(6,4,4)
spec_root0(10,4,8)
```

`spec_root1` *Give the spectral parameters for eigenvalues of mc-matrices equal to one*

Description

Give the spectral parameters for eigenvalues of mc-matrices equal to one.

Usage

```
spec_root1(mo, root1 = numeric(0), iorder = 0, siorder = 0)
```

Arguments

`mo`	mc order.
`root1`	Jordan block lengths for the unit roots, a vector of positive integer numbers.
`iorder`	order of integration, a non-negative integer.
`siorder`	order of seasonal integration, a non-negative integer.

Details

The specifications given by `root1`, `iorder` and `siorder` are combined and the spectral parameters prepared.

In principle, argument `root1` is sufficient, the other two are for convenient specification of integration and seasonal integration.

TODO: rename argument `root1`!

Value

a list with the following components:

`mo`	multi-companion order
`ev.type`	type of the eigenvalues
`co.type`	not used currently (**:todo:**)
`n.root`	number of non-zero roots

ev.abs	absolute values of roots
ev.arg	arguments of eigenvalues (0 for positive ev)
block.length	lengths of Jordan blocks
co.abs	absolute values of seed parameters
co.arg	arguments of seed parameters (Hz: 0 for positive; 1/2 for negative)
co1	temporary hack; **TODO:** check the calling code and remove it!

Author(s)

Georgi N. Boshnakov

See Also

mcSpec, spec_root0

Examples

```
spec_root1(4, root1 = 1)
spec_root1(4, root1 = c(1,0,0,0))  # same
spec_root1(4, iorder = 1)          # same

spec_root1(4, root1 = 2)
spec_root1(4, root1 = c(2,0,0,0))  # same
spec_root1(4, iorder = 2)          # same

spec_root1(4, root1 = c(1,1,1,1))
spec_root1(4, siorder = 1)          # same

spec_root1(4, root1 = c(2,2,2,2))
spec_root1(4, siorder = 2)          # same


spec_root1(4, root1 = c(2,1,1,1))
spec_root1(4, iorder = 1, siorder = 1) # same


spec_root1(4, root1 = c(2,1))
spec_root1(4, root1 = c(2,1,1))
```

---

spec_seeds1 *Generate seed parameters for unit mc-eigenvectors*

Description

Generates seed parameters for mc-eigenvectors corresponding to unit roots.

Usage

```
spec_seeds1(len.block, mo)
```

Arguments

`len.block`	lengths of Jordan blocks, a vector of positive integers.
`mo`	multi-companion order.

Details

Creates a matrix of seed parameters corresponding to unit eigenvalues of a multi-companion matrix of multi-companion order `mo`. `len.block` gives the sizes of the Jordan blocks corresponding to eigenvalues equal to one.

In general, the entries are filled with NA's but for some configurations some (or even all) of the entries are uniquely determined up to a linear transformation. In such cases a "canonical" choice is made.

The generated seed parameters can be considered to be `"top"` or `"bottom"`, as needed. (**TODO:** check this claim, I have forgotten the details but think that this is the reason that it is not necessary to have an argument for the dimension of the matrix).

`spec_seeds1` can be used by model fitting functions to prepare parameters for estimation but see `spec_root1` and `mcSpec` for a more comprehensive treatment.

Value

a matrix with `mo` rows and `sum(len.block)` columns

Note

TODO: the treatment of "canonical" cases is incomplete, see also the comments in the source code of the function.

TODO: explain the `Inf` and `-Inf` output entries for some configurations (e.g. the last example below).

"co" in the name of `spec_seeds1` is short for coefficient.

Author(s)

Georgi N. Boshnakov

See Also

`spec_root1`, `mcSpec`

Examples

```
spec_seeds1(c(1), mo = 4)       # NA's
spec_seeds1(c(1,1), mo = 4)     # NA's
spec_seeds1(c(1,1,1), mo = 4)   # NA's (but for parameterisation
                          #     a different approach is used)
spec_seeds1(c(1,1,1,1), mo = 4) # identity matrix but other bases are good too
spec_seeds1(c(2,2,2,2), mo = 4) # no NA's, tops of gen.evecs can be chosen 0
spec_seeds1(c(2,1,1,1), mo = 4) # (can be improved)
spec_seeds1(c(2,1), mo = 4)     # NA's
```

`VAR2pcfilter` *PAR representations of VAR models*

Description

Give the univariate periodic autoregression representation of a VAR model. Several arrangements are supported as discussed by Boshnakov and Iqelan (2009). If the VAR model contains unit roots on the unit circle, then the univariate model is periodically integrated.

Usage

```
VAR2pcfilter(phi, ..., Sigma, Phi0, Phi0inv, D, what = "coef", perm)
```

Arguments

`phi`	VAR coefficients, a matrix, see Details.
`...`	alternative way to specify the VAR coefficients by giving a matrix for each lag in separate arguments, see section 'Details'.
`Sigma`	covariance matrix of innovations.
`Phi0`	coefficient matrix at lag 0 (alternative to `Sigma`).
`Phi0inv`	inverse of `Phi0` (alternative to `Sigma` and `Phi0`). If `Phi0inv` is lower triangular, then it is the Cholesky factor of Sigma (in Sigma $= LDL'$).
`D`	the diagonal matrix corresponding to `Phi0`, not used if `Sigma` is specified.
`what`	what to return, a string. If equal to `"coef"`, return the PAR coefficients only (as a matrix with one row for each "season"); if equal to `"coef.and.var"` return also the innovation variances. Otherwise return additional quantities (useful for exploration).
`perm`	a permutation specifying the ordering of the variables when treated as "seasons". The default, `d:1`, corresponds to the U-form, see section 'Details'.

Details

VAR2pcfilter converts a VAR model to a scalar periodic autoregressive (PAR) model. There are various ways to specify a VAR model and associate its variables with seasons of the scalar representation, see Boshnakov and Iqelan (2009) for a detailed discussion and the terminology used here.

The VAR coefficients phi,... are those in the standard form of the VAR model (e.g., see Boshnakov and Iqelan 2009). There are two ways to specify them. The first is to put them side by side in a matrix $[\Phi_1, \ldots, \Phi_p]$ and give this matrix as argument phi. Alternatively, the matrices Φ_i may be given directly as arguments to VAR2pcfilter, as in VAR2pcfilter(Phi1, Phi2, Phi3, Sigma = Sigma).

The specification of the model can be completed by giving the covariance matrix, Sigma, of the innovations. Alternatively, it is possible to give the components of the UDU' decomposition of Sigma. In this case argument D is a vector giving the diagonal of the matrix D, while Phi0inv represents the upper triangular matrix U. A further option is to use argument Phi0 to specify the inverse of U. In summary, give either Sigma or D and one of Phi0inv and Phi0.

Phi0 can e interpreted as the coefficient at lag zero in the U-form (Boshnakov and Iqelan 2009) of the VAR model. diag(D) is the variance matrix of the innovations in that form. D also gives the variances of the innovations in the PAR (periodic autoregression) form.

By default, VAR2pcfilter constructs the U-form of the VAR model and extracts the coefficients of the PAR filter from it. This means that the variables in the multivariate vector are given "seasons" in reverse order (the first variable takes the last season, and so on). For the reasons behind this default, see Boshnakov and Iqelan (2009). Another arrangement can be chosen with the help of argument perm. perm should be a permutation specifying the desired allocation of variables to seasons. The default corresponds to perm=d:1, where d is the number of seasons. perm=1:d could be used to request the "natural" order.

When D and Phi0inv (or Phi0) are given, the matrix Sigma is not computed if argument perm is missing but it is if perm is present. This means that perm = d:1 may be used to force the formation of Sigma and recomputation of Phi0 and Phi0inv. This is redundant if the latter two are unit upper-triangular (which is assumed but not checked) but may be handy if, for example, the Cholesky decomposition with a lower triangular matrix is available.

Value

If what="coef", a matrix containing the periodic model coefficients (one row for each season).

If what="coef.and.var", a list containing the coefficients and the innovations' variances:

pcfilter	PAR coefficients, a matrix
var	innovation variances, a vector

Otherwise the returned list contains an additional component, Uform, which is itself a list with components:

Sigma	covariance matrix of innovations,
U0	coefficient for lag zero,
U	the remaining AR coefficients,
U0inv	the inverse of U0,

`perm` permutation giving the season of each variable.

Note: `U0` and `U` correspond to $A0$ and A in the reference (Boshnakov and Iqelan 2009).

Note

This function uses some non-exported internal functions:

.ldl Computes the LDL' Cholesky decomposition with unit lower-triangular matrix L,

.udu Computes the UDU' Cholesky decomposition with unit upper-triangular matrix U.

Could export these if they are deemed more widely useful.

Author(s)

Georgi N. Boshnakov

References

Boshnakov GN, Iqelan BM (2009). "Generation of time series models with given spectral properties." *J. Time Series Anal.*, **30**(3), 349–368. ISSN 0143-9782, doi:10.1111/j.14679892.2009.00617.x.

See Also

`mf_VSform`, `sim_pcfilter`

Examples

```
## create a pc filter
rfi <- sim_pcfilter(2,3)
rfi$pcfilter

## turn it into VAR form
flt <- new("MultiFilter", coef = rfi$pcfilter)
I1 <- mf_VSform(flt, form="I")
I1

## from VAR to scalar form
flt2 <- VAR2pcfilter(I1$Phi, Sigma = I1$Phi0inv %*% t(I1$Phi0inv))
flt2

## confirm that we are back to the original
##   (VAR2pcfilter doesn't drop redundant zeroes, so we do it manually)
    all.equal(flt2[ , 1:3], rfi$pcfilter) ## TRUE
```

Index