Package ‘fastmatrix’

September 5, 2020

Type Package

Title Fast Computation of some Matrices Useful in Statistics

Version 0.2-35

Date 2020-09-03

Author Felipe Osorio [aut, cre] (<https://orcid.org/0000-0002-4675-5201>), Alonso Ogueda [aut]

Maintainer Felipe Osorio <felipe.osorios@usm.cl>

Description Small set of functions to fast computation of some matrices and operations useful in statistics.

Depends R(>= 3.5.0)

License GPL-3

URL https://faosorios.github.io/fastmatrix/

NeedsCompilation yes

LazyLoad yes

Repository CRAN

Date/Publication 2020-09-05 19:40:03 UTC

R topics documented:

`array.mult` *Array multiplication*

Description

Multiplication of 3-dimensional arrays was first introduced by Bates and Watts (1980). More extensions and technical details can be found in Wei (1998).

Usage

```
array.mult(a, b, x)
```

Arguments

`a`	a numeric matrix.
`b`	a numeric matrix.
`x`	a three-dimensional array.

Details

Let $\mathbf{X} = (x_{tij})$ be a 3-dimensional $n \times p \times q$ where indices t, i and j indicate face, row and column, respectively. The product $\mathbf{Y} = \mathbf{AXB}$ is an $n \times r \times s$ array, with $\mathbf{A}$ and $\mathbf{B}$ are $r \times p$ and $q \times s$ matrices respectively. The elements of $\mathbf{Y}$ are defined as:

$$y_{tkl} = \sum_{i=1}^{p} \sum_{j=1}^{q} a_{ki} x_{tij} b_{jl}$$

Value

`array.mult` returns a 3-dimensional array of dimension $n \times r \times s$.

References

Bates, D.M., Watts, D.G. (1980). Relative curvature measures of nonlinearity. *Journal of the Royal Statistical Society, Series B* **42**, 1-25.

Wei, B.C. (1998). *Exponential Family Nonlinear Models*. Springer, New York.

See Also

array, matrix, bracket.prod.

Examples

```
x <- array(0, dim = c(2,3,3)) # 2 x 3 x 3 array
x[,,1] <- c(1,2,2,4,3,6)
x[,,2] <- c(2,4,4,8,6,12)
x[,,3] <- c(3,6,6,12,9,18)

a <- matrix(1, nrow = 2, ncol = 3)
b <- matrix(1, nrow = 3, ncol = 2)

y <- array.mult(a, b, x) # a 2 x 2 x 2 array
y
```

`bracket.prod` *Bracket product*

Description

Bracket product of a matrix and a 3-dimensional array.

Usage

```
bracket.prod(a, x)
```

Arguments

a	a numeric matrix.
x	a three-dimensional array.

Details

Let $\mathbf{X} = (x_{tij})$ be a 3-dimensional $n \times p \times q$ array and $\mathbf{A}$ an $m \times n$ matrix, then $\mathbf{Y} = [\mathbf{A}][\mathbf{X}]$ is called the bracket product of $\mathbf{A}$ and $\mathbf{X}$, that is an $m \times p \times q$ with elements

$$y_{tij} = \sum_{k=1}^{n} a_{tk} x_{kij}$$

Value

`bracket.prod` returns a 3-dimensional array of dimension $m \times p \times q$.

References

Wei, B.C. (1998). *Exponential Family Nonlinear Models*. Springer, New York.

See Also

array, matrix, array.mult.

Examples

```
x <- array(0, dim = c(2,3,3)) # 2 x 3 x 3 array
x[,,1] <- c(1,2,2,4,3,6)
x[,,2] <- c(2,4,4,8,6,12)
x[,,3] <- c(3,6,6,12,9,18)

a <- matrix(1, nrow = 3, ncol = 2)

y <- bracket.prod(a, x) # a 3 x 3 x 3 array
y
```

---

comm.info *Compact information to construct the commutation matrix*

Description

This function provides the minimum information required to create the commutation matrix.

The commutation matrix is a square matrix of order mn that, for an $m \times n$ matrix $\mathbf{A}$, transform vec$(\mathbf{A})$ to vec$(\mathbf{A}^T)$.

Usage

```
comm.info(m = 1, n = m, condensed = TRUE)
```

Arguments

m	a positive integer row dimension.
n	a positive integer column dimension.
condensed	logical. Information should be returned in compact form?

Details

This function returns a list containing two vectors that represent an element of the commutation matrix and is accesed by the indexes in vectors row and col. This information is used by function comm.prod to do some operations involving the commutation matrix without forming it. This information also can be obtained using function commutation.

Value

A list containing the following elements:

`row`	vector of indexes, each entry represents the row index of the commutation matrix.
`col`	vector of indexes, each entry represents the column index of the commutation matrix. Only present if `condensed = FALSE`.
`m`	positive integer, row dimension.
`n`	positive integer, column dimension.

References

Magnus, J.R., Neudecker, H. (1979). The commutation matrix: some properties and applications. *The Annals of Statistics* **7**, 381-394.

See Also

`commutation`, `comm.prod`

Examples

```
z <- comm.info(m = 3, n = 2, condensed = FALSE)
z # where are the ones in commutation matrix of order '3,2'?

K32 <- commutation(m = 3, n = 2, matrix = TRUE)
K32 # only recommended if m and n are very small
```

`comm.prod` *Matrix multiplication envolving the commutation matrix*

Description

Given the row and column dimension of a commutation and matrix `x`, performs one of the matrix-matrix operations:

- $\mathbf{Y} = \mathbf{KX}$, if `side = "left"` and `transposed = FALSE`, or
- $\mathbf{Y} = \mathbf{K}^T\mathbf{X}$, if `side = "left"` and `transposed = TRUE`, or
- $\mathbf{Y} = \mathbf{XK}$, if `side = "right"` and `transposed = FALSE`, or
- $\mathbf{Y} = \mathbf{XK}^T$, if `side = "right"` and `transposed = TRUE`,

where $\mathbf{K}$ is the commutation matrix of order mn. The main aim of `comm.prod` is to do this matrix multiplication **without forming** the commutation matrix.

Usage

```
comm.prod(m = 1, n = m, x = NULL, transposed = FALSE, side = "left")
```

Arguments

`m`	a positive integer row dimension.
`n`	a positive integer column dimension.
`x`	numeric matrix (or vector).
`transposed`	logical. Commutation matrix should be transposed?
`side`	a string selecting if commutation matrix is pre-multiplying `x`, that is `side = "left"` or post-multiplying `x`, by using `side = "right"`.

Details

Underlying `Fortran` code only uses information provided by `comm.info` to performs the matrix multiplication. The commutation matrix is **never** created.

See Also

`commutation`

Examples

```
K42 <- commutation(m = 4, n = 2, matrix = TRUE)
x <- matrix(1:24, ncol = 3)
y <- K42 %*% x

z <- comm.prod(m = 4, n = 2, x) # K42 is not stored
all(z == y) # matrices y and z are equal!
```

`commutation` *Commutation matrix*

Description

This function returns the commutation matrix of order mn which transforms, for an $m \times n$ matrix $\mathbf{A}$, $\text{vec}(\mathbf{A})$ to $\text{vec}(\mathbf{A}^T)$.

Usage

```
commutation(m = 1, n = m, matrix = FALSE, condensed = FALSE)
```

Arguments

`m`	a positive integer row dimension.
`n`	a positive integer column dimension.
`matrix`	a logical indicating whether the commutation matrix will be returned.
`condensed`	logical. Information should be returned in compact form?

Details

This function is a wrapper function for the function `comm.info`. This function provides the minimum information required to create the commutation matrix. If option `matrix = FALSE` the commutation matrix is stored in two vectors containing the coordinate list of indexes for rows and columns. Option `condensed = TRUE` only returns vector of indexes for the rows of commutation matrix.

Warning: `matrix = TRUE` is **not** recommended, unless the order `m` **and** `n` be small. This matrix can require a huge amount of storage.

Value

Returns an mn by mn matrix (if requested).

References

Magnus, J.R., Neudecker, H. (1979). The commutation matrix: some properties and applications. *The Annals of Statistics* **7**, 381-394.

Magnus, J.R., Neudecker, H. (2007). *Matrix Differential Calculus with Applications in Statistics and Econometrics*, 3rd Edition. Wiley, New York.

See Also

`comm.info`

Examples

```
z <- commutation(m = 100, condensed = TRUE)
object.size(z) # 40.6 Kb of storage

z <- commutation(m = 100, condensed = FALSE)
object.size(z) # 80.7 Kb of storage

K100 <- commutation(m = 100, matrix = TRUE) # time: < 2 secs
object.size(K100) # 400 Mb of storage, do not request this matrix!

# a small example
K32 <- commutation(m = 3, n = 2, matrix = TRUE)
a <- matrix(1:6, ncol = 2)
v <- K32 %*% vec(a)
all(vec(t(a)) == as.vector(v)) # vectors are equal!
```

`dupl.cross` *Matrix crossproduct envolving the duplication matrix*

Description

Given the order of two duplication matrices and matrix `x`, this function performs the operation: $\mathbf{Y} = \mathbf{D}_n^T \mathbf{X} \mathbf{D}_k$, where $\mathbf{D}_n$ and $\mathbf{D}_k$ are duplication matrices of order n and k, respectively.

Usage

```
dupl.cross(n = 1, k = n, x = NULL)
```

Arguments

n	order of the duplication matrix used pre-multiplying x.
k	order of the duplication matrix used post-multiplying x. By default k = n is used.
x	numeric matrix, this argument is required.

Details

This function calls dupl.prod to performs the matrix multiplications required but **without forming** any duplication matrices.

See Also

dupl.prod

Examples

```
D2 <- duplication(n = 2, matrix = TRUE)
D3 <- duplication(n = 3, matrix = TRUE)
x <- matrix(1, nrow = 9, ncol = 4)
y <- t(D3) %*% x %*% D2

z <- dupl.cross(n = 3, k = 2, x) # D2 and D3 are not stored
all(z == y) # matrices y and z are equal!

x <- matrix(1, nrow = 9, ncol = 9)
z <- dupl.cross(n = 3, x = x) # same matrix is used to pre- and post-multiplying x
z # print result
```

---

dupl.info *Compact information to construct the duplication matrix*

Description

This function provides the minimum information required to create the duplication matrix.

Usage

```
dupl.info(n = 1, condensed = TRUE)
```

Arguments

n	order of the duplication matrix.
condensed	logical. Information should be returned in compact form?

Details

This function returns a list containing two vectors that represent an element of the duplication matrix and is accesed by the indexes in vectors `row` and `col`. This information is used by function `dupl.prod` to do some operations involving the duplication matrix without forming it. This information also can be obtained using function `duplication`

Value

A list containing the following elements:

`row`	vector of indexes, each entry represents the row index of the duplication matrix. Only present if `condensed = FALSE`.
`col`	vector of indexes, each entry represents the column index of the duplication matrix.
`order`	order of the duplication matrix.

See Also

`duplication`, `dupl.prod`

Examples

```
z <- dupl.info(n = 3, condensed = FALSE)
z # where are the ones in duplication of order 3?

D3 <- duplication(n = 3, matrix = TRUE)
D3 # only recommended if n is very small
```

`dupl.prod` *Matrix multiplication envolving the duplication matrix*

Description

Given the order of a duplication and matrix `x`, performs one of the matrix-matrix operations:

- $\mathbf{Y} = \mathbf{D}\mathbf{X}$, if `side = "left"` and `transposed = FALSE`, or
- $\mathbf{Y} = \mathbf{D}^T\mathbf{X}$, if `side = "left"` and `transposed = TRUE`, or
- $\mathbf{Y} = \mathbf{X}\mathbf{D}$, if `side = "right"` and `transposed = FALSE`, or
- $\mathbf{Y} = \mathbf{X}\mathbf{D}^T$, if `side = "right"` and `transposed = TRUE`,

where $\mathbf{D}$ is the duplication matrix of order n. The main aim of `dupl.prod` is to do this matrix multiplication **without forming** the duplication matrix.

Usage

```
dupl.prod(n = 1, x, transposed = FALSE, side = "left")
```

Arguments

`n`	order of the duplication matrix.
`x`	numeric matrix (or vector).
`transposed`	logical. Duplication matrix should be transposed?
`side`	a string selecting if duplication matrix is pre-multiplying `x`, that is `side = "left"` or post-multiplying `x`, by using `side = "right"`.

Details

Underlying `C` code only uses information provided by `dupl.info` to performs the matrix multiplication. The duplication matrix is **never** created.

See Also

`duplication`

Examples

```
D4 <- duplication(n = 4, matrix = TRUE)
x <- matrix(1, nrow = 16, ncol = 2)
y <- crossprod(D4, x)

z <- dupl.prod(n = 4, x, transposed = TRUE) # D4 is not stored
all(z == y) # matrices y and z are equal!
```

`duplication` *Duplication matrix*

Description

This function returns the duplication matrix of order n which transforms, for a symmetric matrix $\mathbf{A}$, $\mathsf{vech}(\mathbf{A})$ into $\mathsf{vec}(\mathbf{A})$.

Usage

```
duplication(n = 1, matrix = FALSE, condensed = FALSE)
```

Arguments

`n`	order of the duplication matrix.
`matrix`	a logical indicating whether the duplication matrix will be returned.
`condensed`	logical. Information should be returned in compact form?.

Details

This function is a wrapper function for the function `dupl.info`. This function provides the minimum information required to create the duplication matrix. If option `matrix = FALSE` the duplication matrix is stored in two vectors containing the coordinate list of indexes for rows and columns. Option `condensed = TRUE` only returns vector of indexes for the columns of duplication matrix.

Warning: `matrix = TRUE` is **not** recommended, unless the order `n` be small. This matrix can require a huge amount of storage.

Value

Returns an n^2 by $n(n+1)/2$ matrix (if requested).

References

Magnus, J.R., Neudecker, H. (1980). The elimination matrix, some lemmas and applications. *SIAM Journal on Algebraic Discrete Methods* **1**, 422-449.

Magnus, J.R., Neudecker, H. (2007). *Matrix Differential Calculus with Applications in Statistics and Econometrics*, 3rd Edition. Wiley, New York.

See Also

`dupl.info`

Examples

```
z <- duplication(n = 100, condensed = TRUE)
object.size(z) # 40.5 Kb of storage

z <- duplication(n = 100, condensed = FALSE)
object.size(z) # 80.6 Kb of storage

D100 <- duplication(n = 100, matrix = TRUE)
object.size(D100) # 202 Mb of storage, do not request this matrix!

# a small example
D3 <- duplication(n = 3, matrix = TRUE)
a <- matrix(c( 1, 2, 3,
               2, 3, 4,
               3, 4, 5), nrow = 3)
upper <- vech(a)
v <- D3 %*% upper
all(vec(a) == as.vector(v)) # vectors are equal!
```

equilibrate — *Column equilibration of a rectangular matrix*

Description

scale is generic function whose default method centers and/or scales the columns of a numeric matrix.

Usage

```
equilibrate(x, scale = TRUE)
```

Arguments

x	a numeric matrix.
scale	a logical value, the columns of x must be scaled to norm unity?.

Value

For scale = TRUE, the equilibrated (each column scaled to norm one) matrix. The scalings and an approximation of the reciprocal condition number, are returned as attributes "scales" and "condition".

Examples

```
x <- matrix(c(1, 1, 1,
              1, 2, 1,
              1, 3, 1,
              1, 1,-1,
              1, 2,-1,
              1, 3,-1), ncol = 3, byrow = TRUE)
x <- equilibrate(x)
apply(x, 2, function(x) sum(x^2)) # all 1
```

hadamard — *Hadamard product of two matrices*

Description

This function returns the Hadamard or element-wise product of two matrices x and y, that have the same dimensions.

Usage

```
hadamard(x, y = x)
```

Arguments

x	a numeric matrix or vector.
y	a numeric matrix or vector.

Value

A matrix with the same dimension of x (and y) which corresponds to the element-by-element product of the two matrices.

References

Styan, G.P.H. (1973). Hadamard products and multivariate statistical analysis, *Linear Algebra and Its Applications* **6**, 217-240.

Examples

```
x <- matrix(rep(1:10, times = 5), ncol = 5)
y <- matrix(rep(1:5, each = 10), ncol = 5)
z <- hadamard(x, y)
z
```

---

matrix.inner *Compute the inner product between two rectangular matrices*

Description

Computes the inner product between two rectangular matrices calling BLAS.

Usage

```
matrix.inner(x, y = x)
```

Arguments

x	a numeric matrix.
y	a numeric matrix.

Value

a real value, indicating the inner product between two matrices.

Examples

```
x <- matrix(c(1, 1, 1,
              1, 2, 1,
              1, 3, 1,
              1, 1,-1,
              1, 2,-1,
              1, 3,-1), ncol = 3, byrow = TRUE)
y <- matrix(1, nrow = 6, ncol = 3)
matrix.inner(x, y)

# must be equal
matrix.norm(x, type = "Frobenius")^2
matrix.inner(x)
```

`matrix.norm` *Compute the norm of a rectangular matrix*

Description

Computes a matrix norm of `x` using LAPACK. The norm can be the one (`"1"`) norm, the infinity (`"inf"`) norm, the Frobenius norm, the maximum modulus (`"maximum"`) among elements of a matrix, as determined by the value of `type`.

Usage

```
matrix.norm(x, type = "Frobenius")
```

Arguments

`x`	a numeric matrix.
`type`	character string, specifying the *type* of matrix norm to be computed. A character indicating the type of norm desired. `"1"` specifies the **one** norm, (maximum absolute column sum); `"Inf"` specifies the **inf**inity norm (maximum absolute row sum); `"Frobenius"` specifies the **Frobenius** norm (the Euclidean norm of `x` treated as if it were a vector); `"maximum"` specifies the **maximum** modulus of all the elements in `x`.

Details

As function `norm` in package **base**, method of `matrix.norm` calls the LAPACK function `dlange`.

Note that the 1-, Inf- and maximum norm is faster to calculate than the Frobenius one.

Value

The matrix norm, a non-negative number.

Examples

```
# a tiny example
x <- matrix(c(1, 1, 1,
              1, 2, 1,
              1, 3, 1,
              1, 1,-1,
              1, 2,-1,
              1, 3,-1), ncol = 3, byrow = TRUE)
matrix.norm(x, type = "Frobenius")
matrix.norm(x, type = "1")
matrix.norm(x, type = "Inf")

# an example not that small
n <- 1000
x <- .5 * diag(n) + 0.5 * matrix(1, nrow = n, ncol = n)
matrix.norm(x, type = "Frobenius")
matrix.norm(x, type = "1")
matrix.norm(x, type = "Inf")
matrix.norm(x, type = "maximum") # equal to 1
```

---

minkowski — *Computes the p-norm of a vector*

Description

Computes a p-norm of vector `x` using BLAS. The norm can be the one (`p = 1`) norm, Euclidean (`p = 2`) norm, the infinity (`p = Inf`) norm. For other values `p >= 1` the underlying Fortran code is based on ideas of BLAS Level 1.

Usage

```
minkowski(x, p = 2)
```

Arguments

`x`	a numeric vector.
`p`	a number, specifying the *type* of norm desired. Possible values include real number greater or equal to 1, or Inf, Default value is `p = 2`.

Details

Method of `minkowski` calls BLAS functions `dasum` (`p = 1`), `dnrm2` (`p = 2`), `idamax` (`p = Inf`). For other values, a Fortran subroutine using unrolled cycles is called.

Value

The vector p-norm, a non-negative number.

Examples

```
# a tiny example
x <- rnorm(1000)
minkowski(x, p = 1)
minkowski(x, p = 1.5)
minkowski(x, p = 2)
minkowski(x, p = Inf)

x <- x / minkowski(x)
minkowski(x, p = 2) # equal to 1
```

`power.method` *Power method to approximate dominant eigenvalue and eigenvector*

Description

The power method seeks to determine the eigenvalue of maximum modulus, and a corresponding eigenvector.

Usage

```
power.method(x, only.value = FALSE, maxiter = 100, tol = 1e-8)
```

Arguments

`x`	a symmetric matrix.
`only.value`	if `TRUE`, only the dominant eigenvalue is returned, otherwise both dominant eigenvalue and eigenvector are returned.
`maxiter`	the maximum number of iterations. Defaults to `100`
`tol`	a numeric tolerance.

Value

When `only.value` is not true, as by default, the result is a list with components `"value"` and `"vector"`. Otherwise only the dominan eigenvalue is returned. The performed number of iterations to reach convergence is returned as attribute `"iterations"`.

See Also

`eigen` for eigenvalues and eigenvectors computation.

Examples

```
n <- 1000
x <- .5 * diag(n) + 0.5 * matrix(1, nrow = n, ncol = n)

# dominant eigenvalue must be (n + 1) / 2
z <- power.method(x, only.value = TRUE)
```

`sherman.morrison` *Sherman-Morrison formula*

Description

The Sherman-Morrison formula gives a convenient expression for the inverse of the rank 1 update $(\mathbf{A} + \mathbf{b}\mathbf{d}^T)$ where $\mathbf{A}$ is a $n \times n$ matrix and $\mathbf{b}$, $\mathbf{d}$ are n-dimensional vectors. Thus

$$(\mathbf{A} + \mathbf{b}\mathbf{d}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{b}\mathbf{d}^T\mathbf{A}^{-1}}{1 + \mathbf{d}^T\mathbf{A}^{-1}\mathbf{b}}.$$

Usage

```
sherman.morrison(a, b, d = b, inverted = FALSE)
```

Arguments

`a`	a numeric matrix.
`b`	a numeric vector.
`d`	a numeric vector.
`inverted`	logical. If `TRUE`, `a` is supposed to contain its *inverse*.

Details

Method of `sherman.morrison` calls BLAS level 2 subroutines `dgemv` and `dger` for computational efficiency.

Value

a square matrix of the same order as `a`.

Examples

```
n <- 10
ones <- rep(1, n)
a <- 0.5 * diag(n)
z <- sherman.morrison(a, ones, 0.5 * ones)
z
```

sweep.operator *Gauss-Jordan sweep operator for symmetric matrices*

Description

Perform the sweep operation (or reverse sweep) on the diagonal elements of a symmetric matrix.

Usage

```
sweep.operator(x, k = 1, reverse = FALSE)
```

Arguments

`x`	a symmetric matrix.
`k`	elements (if `k` is vector) of the diagonal which will be sweeped.
`reverse`	logical. If `reverse = TRUE` the reverse sweep is performed.

Details

The symmetric sweep operator is a powerful tool in computational statistics with uses in stepwise regression, conditional multivariate normal distributions, MANOVA, and more.

Value

a square matrix of the same order as `x`.

References

Goodnight, J.H. (1979). A tutorial on the SWEEP operator. *The American Statistician* **33**, 149-158.

Examples

```
# tiny example of regression, last column contains 'y'
xy <- matrix(c(1, 1, 1, 1,
               1, 2, 1, 3,
               1, 3, 1, 3,
               1, 1,-1, 2,
               1, 2,-1, 2,
               1, 3,-1, 1), ncol = 4, byrow = TRUE)
z <- crossprod(xy)
z <- sweep.operator(z, k = 1:3)
cf <- z[1:3,4] # regression coefficients
RSS <- z[4,4]  # residual sum of squares

# an example not that small
x <- matrix(rnorm(1000 * 100), ncol = 100)
xx <- crossprod(x)
z <- sweep.operator(xx, k = 1)
```

vec *vectorization of a matrix*

Description

This function returns a vector obtained by stacking the columns of x

Usage

```
vec(x)
```

Arguments

`x`	a numeric matrix.

Value

Let `x` be a n by m matrix, then `vec(x)` is a nm-dimensional vector.

Examples

```
x <- matrix(rep(1:10, each = 10), ncol = 10)
x
y <- vec(x)
y
```

vech *vectorization the lower triangular part of a square matrix*

Description

This function returns a vector obtained by stacking the lower triangular part of a square matrix.

Usage

```
vech(x)
```

Arguments

`x`	a square matrix.

Value

Let `x` be a n by n matrix, then `vech(x)` is a $n(n+1)/2$-dimensional vector.

Examples

```
x <- matrix(rep(1:10, each = 10), ncol = 10)
x
y <- vech(x)
y
```

Index