Package 'fastmatrix'

August 1, 2022

Type Package

Title Fast Computation of some Matrices Useful in Statistics

Version 0.4-124

Date 2022-07-31

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 and econometrics. Currently, there are functions for efficient
computation of duplication, commutation and symmetrizer matrices with minimal storage
requirements. Some commonly used matrix decompositions (LU and LDL), basic matrix
operations (for instance, Hadamard, Kronecker products and the Sherman-Morrison formula)
and iterative solvers for linear systems are also available. In addition, the package
includes a number of common statistical procedures such as the sweep operator, weighted
mean and covariance matrix using an online algorithm, linear regression (using Cholesky,
QR, SVD, sweep operator and conjugate gradients methods), ridge regression (with optimal
selection of the ridge parameter considering several procedures), functions to compute
the multivariate skewness, kurtosis, Mahalanobis distance (checking the positive
defineteness) and the Wilson-Hilferty transformation of chi squared variables. Furthermore,
the package provides interfaces to C code callable by another C code from other R packages.

Depends R(>= 3.5.0)

License GPL-3

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

NeedsCompilation yes

LazyLoad yes

R topics documented:

Index   **57**

---

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

---

asSymmetric	*Force a matrix to be symmetric*

Description

Force a square matrix x to be symmetric

Usage

```
asSymmetric(x, lower = TRUE)
```

Arguments

x	a square matrix to be forced to be symmetric.
lower	logical, should the upper (lower) triangle be replaced with the lower (upper) triangle?

Value

a square symmetric matrix.

Examples

```
a <- matrix(1:16, ncol = 4)
isSymmetric(a) # FALSE
a <- asSymmetric(a) # copy lower triangle into upper triangle
```

---

bezier   *Computation of Bezier curve*

Description

Computes the Bezier curve based on $n + 1$ control points using the De Casteljau's method.

Usage

```
bezier(x, y, ngrid = 200)
```

Arguments

x, y   vector giving the coordinates of the control points. Missing values are deleted.

ngrid   number of elements in the grid used to compute the smoother.

Details

Given $p_0, p_1, \ldots, p_n$ control points the Bezier curve is given by $B(t)$ defined as

$$B(t) = \left(\begin{array}{c} x(t) \\ y(t) \end{array} \right) = \sum_{k=0}^{n} \binom{n}{k} t^k (1 - t)^k p_k$$

where $t \in [0, 1]$. To evaluate the Bezier curve the De Casteljau's method is used.

Value

A list containing `xgrid` and `ygrid` elements used to plot the Bezier curve.

Examples

```
# a tiny example
x <- c(1.0, 0.25, 1.25, 2.5, 4.00, 5.0)
y <- c(0.5, 2.00, 3.75, 4.0, 3.25, 1.0)
plot(x, y, type = "o")
z <- bezier(x, y, ngrid = 50)
lines(z$xgrid, z$ygrid, lwd = 2, lty = 2, col = "red")

# other simple example
x <- c(4,6,4,5,6,7)
y <- 1:6
plot(x, y, type = "o")
z <- bezier(x, y, ngrid = 50)
lines(z$xgrid, z$ygrid, lwd = 2, lty = 2, col = "red")
```

---

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

---

cg   *Solve linear systems using the conjugate gradients method*

Description

Conjugate gradients (CG) method is an iterative algorithm for solving linear systems with positive definite coefficient matrices.

Usage

```
cg(a, b, maxiter = 200, tol = 1e-7)
```

Arguments

a   a symmetric positive definite matrix containing the coefficients of the linear system.

b   a vector of right-hand sides of the linear system.

maxiter   the maximum number of iterations. Defaults to 200

tol   tolerance level for stopping iterations.

Value

a vector with the approximate solution, the iterations performed are returned as the attribute 'iterations'.

Warning

The underlying C code does not check for symmetry nor positive definitiveness.

References

Golub, G.H., Van Loan, C.F. (1996). *Matrix Computations*, 3rd Edition. John Hopkins University Press.

Hestenes, M.R., Stiefel, E. (1952). Methods of conjugate gradients for solving linear equations. *Journal of Research of the National Bureau of Standards* **49**, 409-436.

See Also

jacobi, seidel, solve

Examples

```
a <- matrix(c(4,3,0,3,4,-1,0,-1,4), ncol = 3)
b <- c(24,30,-24)
z <- cg(a, b)
z # converged in 3 iterations
```

---

cholupdate   *Rank 1 update to Cholesky factorization*

Description

function `chol.update`, where `R = chol(A)` is the original Cholesky factorization of A, returns the upper triangular Cholesky factor of $A + xx^T$, with x a column vector of appropriate dimension.

Usage

```
cholupdate(r, x)
```

Arguments

r   a upper triangular matrix, the Cholesky factor of matrix a.

x   vector defining the rank one update.

References

Golub, G.H., Van Loan, C.F. (2013). *Matrix Computations*, 4th Edition. John Hopkins University Press.

See Also

[chol](#)

Examples

```
a <- matrix(c(1,1,1,1,2,3,1,3,6), ncol = 3)
r <- chol(a)
x <- c(0,0,1)
b <- a + outer(x,x)
r1 <- cholupdate(r, x)
r1
all(r1 == chol(b)) # TRUE
```

---

circulant   *Form a symmetric circulant matrix*

Description

Forms a symmetric circulant matrix using a backwards shift of its first column

Usage

```
circulant(x)
```

Arguments

x	the first column to form the circulant matrix.

Value

A symmetric circulant matrix.

Examples

```
x <- c(2,3,5,7,11,13)
circulant(x)
```

---

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 $\boldsymbol{A}$, transform $\mathsf{vec}(\boldsymbol{A})$ to $\mathsf{vec}(\boldsymbol{A}^T)$.

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:

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

where 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 $\boldsymbol{A}$, $\text{vec}(\boldsymbol{A})$ to $\text{vec}(\boldsymbol{A}^T)$.

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!
```

---

corAR1   *AR(1) Correlation Structure*

Description

This function is a constructor for the `corAR1` correlation matrix representing an autocorrelation structure of order 1.

Usage

```
corAR1(rho, p = 2)
```

Arguments

rho   the value of the lag 1 autocorrelation, which must be between -1 and 1.

p   dimension of the requested correlation matrix.

Value

Returns a p by p matrix.

Examples

```
R <- corAR1(rho = 0.8, p = 5)
```

---

corCS	*Compound Symmetry Correlation Structure*

Description

This function is a constructor for the corCS correlation matrix representing a compound symmetry structure corresponding to uniform correlation.

Usage

```
corCS(rho, p = 2)
```

Arguments

rho   the value of the correlation between any two correlated observations, which must be between -1 and 1.

p   dimension of the requested correlation matrix.

Value

Returns a p by p matrix.

Examples

```
R <- corCS(rho = 0.8, p = 5)
```

---

cov.MSSD	*Mean Square Successive Difference (MSSD) estimator of the covariance matrix*

Description

Returns a list containing the mean and covariance matrix of the data.

Usage

```
cov.MSSD(x)
```

Arguments

x   a matrix or data frame. As usual, rows are observations and columns are variables.

Details

This procedure uses the Holmes-Mergen method using the difference between each successive pairs of observations also known as Mean Square Successive Method (MSSD) to estimate the covariance matrix.

Value

A list containing the following named components:

mean   an estimate for the center (mean) of the data.

cov   the estimated covariance matrix.

References

Holmes, D.S., Mergen, A.E. (1993). Improving the performance of the T^2 control chart. *Quality Engineering* **5**, 619-625.

See Also

[cov](#) and [var](#).

Examples

```
x <- cbind(1:10, c(1:3, 8:5, 8:10))
z0 <- cov(x)
z0
z1 <- cov.MSSD(x)
z1
```

---

cov.weighted   *Weighted covariance matrices*

Description

Returns a list containing estimates of the weighted mean and covariance matrix of the data.

Usage

```
cov.weighted(x, weights = rep(1, nrow(x)))
```

Arguments

x   a matrix or data frame. As usual, rows are observations and columns are variables.

weights   a non-negative and non-zero vector of weights for each observation. Its length must equal the number of rows of x.

Details

The covariance matrix is divided by the number of observations, which arise for instance, when we use the class of elliptical contoured distributions. This differs from the behaviour of function cov.wt.

Value

A list containing the following named components:

mean	an estimate for the center (mean) of the data.
cov	the estimated (weighted) covariance matrix.

References

Clarke, M.R.B. (1971). Algorithm AS 41: Updating the sample mean and dispersion matrix. *Applied Statistics* **20**, 206-209.

See Also

cov.wt, cov and var.

Examples

```
x <- cbind(1:10, c(1:3, 8:5, 8:10))
z0 <- cov.weighted(x) # all weights are 1
D2 <- Mahalanobis(x, center = z0$mean, cov = z0$cov)
p <- ncol(x)
wts <- (p + 1) / (1 + D2) # nice weights!
z1 <- cov.weighted(x, weights = wts)
z1
```

---

dupl.cross	*Matrix crossproduct envolving the duplication matrix*

Description

Given the order of two duplication matrices and matrix x, this function performs the operation: $Y = D_n^T X D_k$, where D_n and 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:

- $Y = DX$, if side = "left" and transposed = FALSE, or
- $Y = D^T X$, if side = "left" and transposed = TRUE, or
- $Y = XD$, if side = "right" and transposed = FALSE, or
- $Y = XD^T$, if side = "right" and transposed = TRUE,

where 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 A, vech(A) into vec(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 | *Equilibration of a rectangular or symmetric matrix* |

Description

Equilibrate a rectangular or symmetric matrix using 2-norm.

Usage

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

Arguments

x   a numeric matrix.

scale   a logical value, x must be scaled to norm unity?

Value

For scale = TRUE, the equilibrated matrix. The scalings and an approximation of the condition number, are returned as attributes "scales" and "condition". If x is a rectangular matrix, only the columns are equilibrated.

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)
z <- equilibrate(x)
apply(z, 2, function(x) sum(x^2)) # all 1

xx <- crossprod(x)
equilibrate(xx)
```

---

frank   *Frank matrix*

Description

This function returns the Frank matrix of order n.

Usage

```
frank(n = 1)
```

Arguments

n   order of the Frank matrix.

Details

A Frank matrix of order n is a square matrix $\boldsymbol{F}_n = (f_{ij})$ defined as

$$f_{ij} = \begin{cases} n - j + 1, & i \leq j, \\ n - j, & i = j + 1, \\ 0, & i \geq j + 2 \end{cases}$$

Value

Returns an n by n matrix.

References

Frank, W.L. (1958). Computing eigenvalues of complex matrices by determinant evaluation and by methods of Danilewski and Wielandt. *Journal of the Society for Industrial and Applied Mathematics* **6**, 378-392.

Examples

```
F5 <- frank(n = 5)
det(F5) # equals 1
```

---

geomean   *Geometric mean*

Description

It calculates the geometric mean using a Fused-Multiply-and-Add (FMA) compensated scheme for accurate computation of floating-point product.

Usage

```
geomean(x)
```

Arguments

x   a numeric vector containing the sample observations.

Details

If x contains any non-positive values, geomean returns NA and a warning message is displayed.

The geometric mean is a measure of central tendency, which is defined as

$$G = \sqrt[n]{x_1 x_2 \ldots x_n} = \Big(\prod_{i=1}^{n} x_i \Big)^{1/n}.$$

This procedure calculates the product required in the geometric mean safely using a compensated scheme as proposed by Graillat (2009).

Value

The geometric mean of the sample, a non-negative number.

References

Graillat, S. (2009). Accurate floating-point product and exponentiation. *IEEE Transactions on Computers* **58**, 994-1000.

Oguita, T., Rump, S.M., Oishi, S. (2005). Accurate sum and dot product. *SIAM Journal on Scientific Computing* **26**, 1955-1988.

See Also

mean, median.

Examples

```
set.seed(149)
x <- rlnorm(1000)
mean(x)    # 1.68169
median(x)  # 0.99663
geomean(x) # 1.01688
```

---

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
```

---

harris.test   *Test for variance homogeneity of correlated variables*

Description

Performs large-sample methods for testing equality of $p \geq 2$ correlated variables.

Usage

```
harris.test(x, test = "Wald")
```

Arguments

x	a matrix or data frame. As usual, rows are observations and columns are variables.
test	test statistic to be used. One of "Wald" (default), "log", "robust" or "log-robust".

Value

A list of class 'harris.test' with the following elements:

statistic	value of the statistic, i.e. the value of either Wald test, using the log-transformation, or distribution-robust versions of the test (robust and log-robust).
parameter	the degrees of freedom for the test statistic, which is chi-square distributed.
p.value	the p-value for the test.
estimate	the estimated covariance matrix.
method	a character string indicating what type of test was performed.

References

Harris, P. (1985). Testing the variance homogeneity of correlated variables. *Biometrika* **72**, 103-107.

Examples

```
x <- iris[,1:4]
z <- harris.test(x, test = "robust")
z
```

---

helmert   *Helmert matrix*

Description

This function returns the Helmert matrix of order n.

Usage

```
helmert(n = 1)
```

Arguments

n   order of the Helmert matrix.

Details

A Helmert matrix of order n is a square matrix defined as

$$\boldsymbol{H}_n = \begin{bmatrix} 1/\sqrt{n} & 1/\sqrt{n} & 1/\sqrt{n} & \dots & 1/\sqrt{n} \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 & \dots & 0 \\ 1/\sqrt{6} & 1/\sqrt{6} & -2/\sqrt{6} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{n(n-1)}} & \frac{1}{\sqrt{n(n-1)}} & \frac{1}{\sqrt{n(n-1)}} & \dots & -\frac{(n-1)}{\sqrt{n(n-1)}} \end{bmatrix}.$$

Helmert matrix is orthogonal and is frequently used in the analysis of variance (ANOVA).

Value

Returns an n by n matrix.

References

Lancaster, H.O. (1965). The Helmert matrices. *The American Mathematical Monthly* **72**, 4-12.

Gentle, J.E. (2007). *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer, New York.

Examples

```
n <- 1000
set.seed(149)
x <- rnorm(n)

H <- helmert(n)
object.size(H) # 7.63 Mb of storage
K <- H[2:n,]
z <- c(K %*% x)
sum(z^2) # 933.1736
```

```
# same that
(n - 1) * var(x)
```

---

is.lower.tri   *Check if a matrix is lower or upper triangular*

Description

Returns TRUE if the given matrix is lower or upper triangular matrix.

Usage

```
is.lower.tri(x, diag = FALSE)
is.upper.tri(x, diag = FALSE)
```

Arguments

x   a matrix of other R object with length(dim(x)) == 2.

diag   logical. Should the diagonal be included?

Value

Check if a matrix is lower or upper triangular. You can also include diagonal to the check.

See Also

[lower.tri](), [upper.tri]()

Examples

```
x <- matrix(rnorm(10 * 3), ncol = 3)
R <- chol(crossprod(x))

is.lower.tri(R)
is.upper.tri(R)
```

---

jacobi   *Solve linear systems using the Jacobi method*

Description

Jacobi method is an iterative algorithm for solving a system of linear equations.

Usage

```
jacobi(a, b, start, maxiter = 200, tol = 1e-7)
```

Arguments

a	a square numeric matrix containing the coefficients of the linear system.
b	a vector of right-hand sides of the linear system.
start	a vector for initial starting point.
maxiter	the maximum number of iterations. Defaults to 200
tol	tolerance level for stopping iterations.

Details

Let D, L, and U denote the diagonal, lower triangular and upper triangular parts of a matrix A. Jacobi's method solve the equation $Ax = b$, iteratively by rewriting $Dx + (L + U)x = b$. Assuming that D is nonsingular leads to the iteration formula

$$x^{(k+1)} = -D^{-1}(L + U)x^{(k)} + D^{-1}b$$

Value

a vector with the approximate solution, the iterations performed are returned as the attribute 'iterations'.

References

Golub, G.H., Van Loan, C.F. (1996). *Matrix Computations*, 3rd Edition. John Hopkins University Press.

See Also

seidel

Examples

```
a <- matrix(c(5,-3,2,-2,9,-1,3,1,-7), ncol = 3)
b <- c(-1,2,3)
start <- c(1,1,1)
z <- jacobi(a, b, start)
z # converged in 15 iterations
```

---

kronecker.prod   *Kronecker product on matrices*

Description

Computes the kronecker product of two matrices, x and y.

Usage

```
kronecker.prod(x, y = x)
```

Arguments

x   a numeric matrix or vector.

y   a numeric matrix or vector.

Details

Let X be an $m \times n$ and Y a $p \times q$ matrix. The $mp \times nq$ matrix defined by

$$
\begin{bmatrix}
x_{11}Y & \dots & x_{1n}Y \\
\vdots & & \vdots \\
x_{m1}Y & \dots & x_{mn}Y
\end{bmatrix},
$$

is called the *Kronecker product* of X and Y.

Value

An array with dimensions `dim(x) * dim(y)`.

References

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

See Also

`kronecker` function from base package is based on `outer`. Our C version is slightly faster.

Examples

```
# block diagonal matrix:
a <- diag(1:3)
b <- matrix(1:4, ncol = 2)
kronecker.prod(a, b)

# examples with vectors
ones <- rep(1, 4)
```

```
y <- 1:3
kronecker.prod(ones, y) # 12-dimensional vector
kronecker.prod(ones, t(y)) # 3 x 3 matrix
```

---

krylov   *Computes a Krylov matrix*

Description

Given A an n by n real matrix and an n-vector b, this function constructs the Krylov matrix K, where

$$K = [b, Ab, \ldots, A^{m-1}b].$$

Usage

```
krylov(a, b, m = ncol(a))
```

Arguments

a   a numeric square matrix of order n by n for which the Krylov matrix is to be computed.

b   a numeric vector of length n.

m   length of the Krylov sequence.

Value

Returns an n by m matrix.

Examples

```
a <- matrix(c(1, 3, 2, -5, 1, 7, 1, 5, -4), ncol = 3, byrow = TRUE)
b <- c(1, 1, 1)
k <- krylov(a, b, m = 4)
k
```

| kurtosis | *Mardia's multivariate skewness and kurtosis coefficients* |

Description

Functions to compute measures of multivariate skewness (b_{1p}) and kurtosis (b_{2p}) proposed by Mardia (1970),

$$b_{1p} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} ((\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T \boldsymbol{S}^{-1} (\boldsymbol{x}_j - \overline{\boldsymbol{x}}))^3,$$

and

$$b_{2p} = \frac{1}{n} \sum_{i=1}^{n} ((\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T \boldsymbol{S}^{-1} (\boldsymbol{x}_j - \overline{\boldsymbol{x}}))^2.$$

Usage

```
kurtosis(x)

skewness(x)
```

Arguments

x   matrix of data with, say, p columns.

References

Mardia, K.V. (1970). Measures of multivariate skewness and kurtosis with applications. *Biometrika* **57**, 519-530.

Mardia, K.V., Zemroch, P.J. (1975). Algorithm AS 84: Measures of multivariate skewness and kurtosis. *Applied Statistics* **24**, 262-265.

Examples

```
setosa <- iris[1:50,1:4]
kurtosis(setosa)
skewness(setosa)
```

---

ldl   *The LDL decomposition*

Description

Compute the LDL decomposition of a real symmetric matrix.

Usage

```
ldl(x)
```

Arguments

x   a symmetric numeric matrix whose LDL decomposition is to be computed.

Value

The factorization has the form $X = LDL^T$, where D is a diagonal matrix and L is unitary lower triangular.

The LDL decomposition of x is returned as a list with components:

lower   the unitary lower triangular factor L.

d   a vector containing the diagonal elements of D.

References

Golub, G.H., Van Loan, C.F. (1996). *Matrix Computations*, 3rd Edition. John Hopkins University Press.

See Also

[chol](#)

Examples

```
a <- matrix(c(2,-1,0,-1,2,-1,0,-1,1), ncol = 3)
z <- ldl(a)
z # information of LDL factorization

# computing det(a)
prod(z$d) # product of diagonal elements of D

# a non-positive-definite matrix
m <- matrix(c(5,-5,-5,3), ncol = 2)
try(chol(m)) # fails
ldl(m)
```

---

lu	*The LU factorization of a square matrix*

Description

lu computes the LU factorization of a matrix.

Usage

```
lu(x)
## Default S3 method:
lu(x)

## S3 method for class 'lu'
solve(a, b, ...)

is.lu(x)
```

Arguments

x	a square numeric matrix whose LU factorization is to be computed.
a	an LU factorization of a square matrix.
b	a vector or matrix of right-hand sides of equations.
...	further arguments passed to or from other methods

Details

The LU factorization plays an important role in many numerical procedures. In particular it is the basic method to solve the equation $Ax = b$ for given matrix A, and vector b.

solve.lu is the method for [solve](#) for lu objects.

is.lu returns TRUE if x is a [list](#) and [inherits](#) from "lu".

Unsuccessful results from the underlying LAPACK code will result in an error giving a positive error code: these can only be interpreted by detailed study of the Fortran code.

Value

The LU factorization of the matrix as computed by LAPACK. The components in the returned value correspond directly to the values returned by DGETRF.

lu	a matrix with the same dimensions as x. The upper triangle contains the U of the decomposition and the strict lower triangle contains information on the L of the factorization.
pivot	information on the pivoting strategy used during the factorization.

Note

To compute the determinant of a matrix (do you *really* need it?), the LU factorization is much more efficient than using eigenvalues (`eigen`). See `det`.

LAPACK uses column pivoting and does not attempt to detect rank-deficient matrices.

References

Anderson. E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. Sorensen, D. (1999). *LAPACK Users' Guide*, 3rd Edition. SIAM.

Golub, G.H., Van Loan, C.F. (1996). *Matrix Computations*, 3rd Edition. John Hopkins University Press.

See Also

`extractL`, `extractU`, `constructX` for reconstruction of the matrices, `lu2inv`

Examples

```
a <- matrix(c(3,2,6,17,4,18,10,-2,-12), ncol = 3)
z <- lu(a)
z # information of LU factorization

# computing det(a)
prod(diag(z$lu)) # product of diagonal elements of U

# solve linear equations
b <- matrix(1:6, ncol = 2)
solve(z, b)
```

---

lu-methods   *Reconstruct the L, U, or X Matrices from an LU object*

Description

Returns the original matrix from which the object was constructed or the components of the factorization.

Usage

```
constructX(x)
extractL(x)
extractU(x)
```

Arguments

x   object representing an LU factorization. This will typically have come from a
                    previous call to `lu`.

Value

constructX returns X, the original matrix from which the lu object was constructed (because of the pivoting the X matrix is not exactly the product between L and U).

extractL returns L. This may be pivoted.

extractU returns U.

See Also

[lu](#).

Examples

```
a <- matrix(c(10,-3,5,-7,2,-1,0,6,5), ncol = 3)
z <- lu(a)
L <- extractL(z)
L
U <- extractU(z)
U
X <- constructX(z)
all(a == X)
```

---

lu2inv   *Inverse from LU factorization*

Description

Invert a square matrix from its LU factorization.

Usage

```
lu2inv(x)
```

Arguments

x   object representing an LU factorization. This will typically have come from a
                   previous call to [lu](#).

Value

The inverse of the matrix whose LU factorization was given.

Unsuccessful results from the underlying LAPACK code will result in an error giving a positive error code: these can only be interpreted by detailed study of the Fortran code.

Source

This is an interface to the LAPACK routine DGETRI. LAPACK is from https://netlib.org/lapack/ and its guide is listed in the references.

References

Anderson. E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. Sorensen, D. (1999). *LAPACK Users' Guide*, 3rd Edition. SIAM.

Golub, G.H., Van Loan, C.F. (1996). *Matrix Computations*, 3rd Edition. John Hopkins University Press.

See Also

lu, solve.

Examples

```
a <- matrix(c(3,2,6,17,4,18,10,-2,-12), ncol = 3)
z <- lu(a)
a %*% lu2inv(z)
```

Mahalanobis   *Mahalanobis distance*

Description

Returns the squared Mahalanobis distance of all rows in x and the vector $\boldsymbol{\mu}$ = center with respect to $\boldsymbol{\Sigma}$ = cov. This is (for vector x) defined as

$$D^2 = (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})$$

Usage

```
Mahalanobis(x, center, cov, inverted = FALSE)
```

Arguments

x	vector or matrix of data. As usual, rows are observations and columns are variables.
center	mean vector of the distribution.
cov	covariance matrix ($p \times p$) of the distribution, must be positive definite.
inverted	logical. If TRUE, cov is supposed to contain the *inverse* of the covariance matrix.

Details

Unlike function mahalanobis, the covariance matrix is factorized using the Cholesky decomposition, which allows to assess if cov is positive definite. Unsuccessful results from the underlying LAPACK code will result in an error message.

See Also

cov, mahalanobis

Examples

```
x <- cbind(1:6, 1:3)
xbar <- colMeans(x)
S <- matrix(c(1,4,4,1), ncol = 2) # is negative definite
D2 <- mahalanobis(x, center = xbar, S)
all(D2 >= 0) # several distances are negative

## next command produces the following error:
## Covariance matrix is possibly not positive-definite
## Not run: D2 <- Mahalanobis(x, center = xbar, S)
```

---

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
```

---

matrix.polynomial   *Evaluates a real general matrix polynomial*

Description

Given $c_0, c_1, \ldots, c_n$ coefficients of the polynomial and $\boldsymbol{A}$ an n by n matrix. This function computes the matrix polynomial

$$\boldsymbol{B} = \sum_{k=0}^{n} c_k \boldsymbol{A}^k,$$

using Horner's scheme, where $\boldsymbol{A}^0 = \boldsymbol{I}$ is the n by n identity matrix.

Usage

```
matrix.polynomial(a, coef = rep(1, power + 1), power = length(coef))
```

Arguments

a	a numeric square matrix of order n by n for which the polinomial is to be computed.
coef	numeric vector containing the coefficients of the polinomial in order of increasing power.
power	a numeric exponent (which is forced to be an integer). If the exponent is zero, a multiple of the identity matrix is returned.

Value

Returns an n by n matrix.

Examples

```
a <- matrix(c(1, 3, 2, -5, 1, 7, 1, 5, -4), ncol = 3, byrow = TRUE)
cf <- c(3, 1, 2)
b <- matrix.polynomial(a, cf)
b # 3 * diag(3) + a + 2 * a %*% a
b <- matrix.polynomial(a, power = 2)
b # diag(3) + a + a %*% a
```

---

mediancenter	*Mediancenter*

Description

It calculates the mediancenter (or geometric median) of multivariate data.

Usage

```
mediancenter(x)
```

Arguments

x   a matrix or data frame. As usual, rows are observations and columns are vari-
                     ables.

Details

The mediancenter for a sample of multivariate observations is computed using a steepest descend
method combined with bisection. The mediancenter invariant to rotations of axes and is useful as a
multivariate generalization of the median of univariate sample.

Value

A list containing the following named components:

median   an estimate for the mediancenter of the data.

iter   the number of iterations performed, it is negative if a degenerate solution is
                    found.

References

Gower, J.C. (1974). Algorithm AS 78: The mediancentre. *Applied Statistics* **23**, 466-470.

See Also

[cov.wt](), [median]().

Examples

```
x <- cbind(1:10, c(1:3, 8:5, 8:10))
z <- mediancenter(x)$median # degenerate solution
xbar <- colMeans(x)
plot(x, xlab = "", ylab = "")
points(x = xbar[1], y = xbar[2], pch = 16, col = "red")
points(x = z[1], y = z[2], pch = 3, col = "blue", lwd = 2)
```

minkowski	*Computes the p-norm of a vector*

Description

Computes a p-norm of vector x. The norm can be the one (p = 1) norm, Euclidean (p = 2) norm, the infinity (p = Inf) norm. The underlying C or Fortran code is inspired 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 for p = Inf calls idamax BLAS function. For other values, C or Fortran subroutines using unrolled cycles are 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
```

moments	*Central moments*

Description

It calculates up to fourth central moments (or moments about the mean), and the skewness and kurtosis coefficients using an online algorithm.

Usage

```
moments(x)
```

Arguments

x   a numeric vector containing the sample observations.

Details

The k-th central moment is defined as

$$m_k = \frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})^k.$$

In particular, the second central moment is the variance of the sample. The sample skewness and kurtosis are defined, respectively, as

$$b_1 = \frac{m_3}{s^3}, \qquad b_2 = \frac{m_4}{s^4} - 3,$$

where s denotes de standard deviation.

Value

A list containing `second`, `third` and `fourth` central moments, and `skewness` and `kurtosis` coefficients.

References

Spicer, C.C. (1972). Algorithm AS 52: Calculation of power sums of deviations about the mean. *Applied Statistics* **21**, 226-227.

See Also

`var`.

Examples

```
set.seed(149)
x <- rnorm(1000)
z <- moments(x)
z
```

---

ols *Fit linear regression model*

Description

Returns an object of class "ols" that represents a linear model fit.

Usage

```
ols(formula, data, subset, na.action, method = "qr", tol = 1e-7, maxiter = 100,
  model = FALSE, x = FALSE, y = FALSE, contrasts = NULL, ...)
```

Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted.
data	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which ols is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset.
method	the least squares fitting method to be used; the options are "cg" (conjugate gradients), "chol", "qr" (the default), "svd" and "sweep".
tol	tolerance for the conjugate gradients (gc) method. Default is tol = 1e-7.
maxiter	The maximum number of iterations for the conjugate gradients (gc) method. Defaults to 100.
model, x, y	logicals. If TRUE the corresponding components of the fit (the model frame, the model matrix, the response) are returned.
contrasts	an optional list. See the contrasts.arg of model.matrix.default.
...	additional arguments (currently disregarded).

Value

ols returns an object of class "ols".

The function summary is used to obtain and print a summary of the results. The generic accessor functions coefficients, fitted.values and residuals extract various useful features of the value returned by ols.

An object of class "ols" is a list containing at least the following components:

coefficients   a named vector of coefficients

residuals	the residuals, that is response minus fitted values.
fitted.values	the fitted mean values.
RSS	the residual sum of squares.
cov.unscaled	a $p \times p$ matrix of (unscaled) covariances of the $\hat{\beta}_j$, $j = 1, \ldots, p$.
call	the matched call.
terms	the [terms](#) object used.
contrasts	(only where relevant) the contrasts used.
y	if requested, the response used.
x	if requested, the model matrix used.
model	if requested (the default), the model frame used.

See Also

[ols.fit](#), [lm](#), [lsfit](#)

Examples

```
# tiny example of regression
y <- c(1, 3, 3, 2, 2, 1)
x <- matrix(c(1, 1,
              2, 1,
              3, 1,
              1,-1,
              2,-1,
              3,-1), ncol = 2, byrow = TRUE)
f0 <- ols(y ~ x) # intercept is included by default
f0 # printing results (QR method was used)

f1 <- ols(y ~ x, method = "svd") # using SVD method instead
f1
```

---

ols.fit   *Fitter Functions for Linear Models*

Description

This function is a *switcher* among various numerical fitting functions ([ols.fit.cg](#), [ols.fit.chol](#), [ols.fit.qr](#), [ols.fit.svd](#) and [ols.fit.sweep](#)). The argument method does the switching: "qr" for [ols.fit.qr](#), etc. This should usually *not* be used directly unless by experienced users.

Usage

```
ols.fit(x, y, method = "qr", tol = 1e-7, maxiter = 100)
```

Arguments

x	design matrix of dimension $n \times q$.
y	vector of observations of length n.
method	currently, methods `"cg"`, `"chol"`, `"qr"` (default), `"svd"` and `"sweep"` are supported.
tol	tolerance for the conjugate gradients (gc) method. Default is `tol = 1e-7`.
maxiter	The maximum number of iterations for the conjugate gradients (gc) method. Defaults to 100.

Value

a [list](#) with components:

coefficients	a named vector of coefficients
residuals	the residuals, that is response minus fitted values.
fitted.values	the fitted mean values.
RSS	the residual sum of squares.
cov.unscaled	a $p \times p$ matrix of (unscaled) covariances of the $\hat{\beta}_j$, $j = 1, \ldots, p$.

See Also

[ols.fit.cg](#), [ols.fit.chol](#), [ols.fit.qr](#), [ols.fit.svd](#), [ols.fit.sweep](#).

Examples

```
set.seed(151)
n <- 100
p <- 2
x <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)
fm <- ols.fit(x = x, y = y, method = "chol")
fm
```

---

ols.fit-methods *Fit a Linear Model*

Description

Fits a linear model, returning the bare minimum computations.

Usage

```
ols.fit.cg(x, y, tol = 1e-7, maxiter = 100)
ols.fit.chol(x, y)
ols.fit.qr(x, y)
ols.fit.svd(x, y)
ols.fit.sweep(x, y)
```

Arguments

x, y	numeric vectors or matrices for the predictors and the response in a linear model. Typically, but not necessarily, x will be constructed by one of the fitting functions.
tol	tolerance for the conjugate gradients (gc) method. Default is tol = 1e-7.
maxiter	The maximum number of iterations for the conjugate gradients (gc) method. Defaults to 100.

Value

The bare bones of an `ols` object: the coefficients, residuals, fitted values, and some information used by `summary.ols`.

See Also

`ols`, `ols.fit`, `lm`

Examples

```
set.seed(151)
n <- 100
p <- 2
x <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)
z <- ols.fit.chol(x, y)
z
```

---

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)
```

---

ridge   *Ridge regression*

Description

Fit a linear model by ridge regression, returning an object of class `"ridge"`.

Usage

```
ridge(formula, data, subset, lambda = 1.0, method = "GCV", ngrid = 200, tol = 1e-07,
  maxiter = 50, na.action, model = FALSE, x = FALSE, y = FALSE, contrasts = NULL, ...)
```

Arguments

formula	an object of class `"`[formula](#)`"` (or one that can be coerced to that class): a symbolic description of the model to be fitted.
data	an optional data frame, list or environment (or object coercible by [as.data.frame](#) to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `ridge` is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the `na.action` setting of [options](#), and is [na.fail](#) if that is unset.
lambda	a scalar or vector of ridge constants. A value of 0 corresponds to ordinary least squares.

method	the method for choosing the ridge parameter lambda. If method = "none", then lambda is 'fixed'. If method = "GCV" (the default) then the ridge parameter is chosen automatically using the generalized cross validation (GCV) criterion. For method = "grid", optimal value of lambda is selected computing the GCV criterion over a grid. If method = "MSE" the optimal ridge parameter is selected minimizing the mean squared estimation error criterion, this is the ORPS1 subroutine by Lee (1987).
ngrid	number of elements in the grid used to compute the GCV criterion. Only required if method = "grid" and lambda is a scalar.
tol	tolerance for the optimization of the GCV criterion. Default is 1e-7.
maxiter	maximum number of iterations. The default is 50.
model, x, y	logicals. If TRUE the corresponding components of the fit (the model frame, the model matrix, the response) are returned.
contrasts	an optional list. See the contrasts.arg of [model.matrix.default]().
...	additional arguments to be passed to the low level regression fitting functions (not implemented).

Details

ridge function fits in linear ridge regression **without** scaling or centering the regressors and the response. In addition, If an intercept is present in the model, its coefficient is penalized.)

Value

A list with the following components:

dims	dimensions of model matrix.
coefficients	a named vector of coefficients.
scale	a named vector of coefficients.
fitted.values	the fitted mean values.
residuals	the residuals, that is response minus fitted values.
RSS	the residual sum of squares.
edf	the effective number of parameters.
GCV	vector (if method = "grid") of GCV values.
HKB	HKB estimate of the ridge constant.
LW	LW estimate of the ridge constant.
lambda	vector (if method = "grid") of lambda values; otherwise, for methods method = "none", "GCV" or "MSE", the value of ridge parameter used by the algorithm.
optimal	value of lambda with the minimum GCV (only relevant if method = "grid").
iterations	number of iterations performed by the algorithm (only relevant if method = "MSE").
call	the matched call.
terms	the [terms]() object used.

contrasts	(only where relevant) the contrasts used.
y	if requested, the response used.
x	if requested, the model matrix used.
model	if requested, the model frame used.

References

Golub, G.H., Heath, M., Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics* **21**, 215-223.

Hoerl, A.E., Kennard, R.W., Baldwin, K.F. (1975). Ridge regression: Some simulations. *Communication in Statistics* **4**, 105-123.

Hoerl, A.E., Kennard, R.W. (1970). Ridge regression: Biased estimation of nonorthogonal problems. *Technometrics* **12**, 55-67.

Lawless, J.F., Wang, P. (1976). A simulation study of ridge and other regression estimators. *Communications in Statistics* **5**, 307-323.

Lee, T.S (1987). Algorithm AS 223: Optimum ridge parameter selection. *Applied Statistics* **36**, 112-118.

See Also

lm, ols

Examples

```
z <- ridge(GNP.deflator ~ ., data = longley, lambda = 4, method = "grid")
z # ridge regression on a grid over seq(0, 4, length = 200)

z <- ridge(GNP.deflator ~ ., data = longley)
z # ridge parameter selected using GCV (default)
```

---

seidel   *Solve linear systems using the Gauss-Seidel method*

Description

Gauss-Seidel method is an iterative algorithm for solving a system of linear equations.

Usage

```
seidel(a, b, start, maxiter = 200, tol = 1e-7)
```

Arguments

a	a square numeric matrix containing the coefficients of the linear system.
b	a vector of right-hand sides of the linear system.
start	a vector for initial starting point.
maxiter	the maximum number of iterations. Defaults to 200
tol	tolerance level for stopping iterations.

Details

Let D, L, and U denote the diagonal, lower triangular and upper triangular parts of a matrix A. Gauss-Seidel method solve the equation $Ax = b$, iteratively by rewriting $(L + D)x + Ux = b$. Assuming that $L + D$ is nonsingular leads to the iteration formula

$$x^{(k+1)} = -(L + D)^{-1}Ux^{(k)} + (L + D)^{-1}b$$

Value

a vector with the approximate solution, the iterations performed are returned as the attribute 'iterations'.

References

Golub, G.H., Van Loan, C.F. (1996). *Matrix Computations*, 3rd Edition. John Hopkins University Press.

See Also

[jacobi](#)

Examples

```
a <- matrix(c(5,-3,2,-2,9,-1,3,1,-7), ncol = 3)
b <- c(-1,2,3)
start <- c(1,1,1)
z <- seidel(a, b, start)
z # converged in 10 iterations
```

---

sherman.morrison   *Sherman-Morrison formula*

Description

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

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

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)
```

---

symm.info   *Compact information to construct the symmetrizer matrix*

Description

This function provides the information required to create the symmetrizer matrix.

Usage

```
symm.info(n = 1)
```

Arguments

n   order of the symmetrizer matrix.

Details

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

Value

A list containing the following elements:

row   vector of indexes, each entry represents the row index of the symmetrizer matrix.
col   vector of indexes, each entry represents the column index of the symmetrizer
                  matrix.
val   vector of values, each entry represents the value of the symmetrizer matrix at
                  element given by row and col indexes.
order   order of the symmetrizer matrix.

See Also

symmetrizer, symm.prod

Examples

```
z <- symm.info(n = 3)
z # elements in symmetrizer matrix of order 3

N3 <- symmetrizer(n = 3, matrix = TRUE)
N3 # only recommended if n is very small
```

---

symm.prod   *Matrix multiplication envolving the symmetrizer matrix*

Description

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

- $Y = NX$, if side = "left", or
- $Y = XN$, if side = "right",

where N is the symmetrizer matrix of order n. The main aim of symm.prod is to do this matrix multiplication **without forming** the symmetrizer matrix.

Usage

```
symm.prod(n = 1, x = NULL, side = "left")
```

Arguments

n	order of the symmetrizer matrix.
x	numeric matrix (or vector).
side	a string selecting if symmetrizer 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 symm.info to performs the matrix multiplication. The symmetrizer matrix is **never** created.

See Also

symmetrizer

Examples

```
N4 <- symmetrizer(n = 4, matrix = TRUE)
x <- matrix(1:32, ncol = 2)
y <- N4 %*% x

z <- symm.prod(n = 4, x) # N4 is not stored
all(z == y) # matrices y and z are equal!
```

symmetrizer	*Symmetrizer matrix*

Description

This function returns the symmetrizer matrix of order n which transforms, for every $n \times n$ matrix A, $\text{vec}(A)$ into $\text{vec}((A + A^T)/2)$.

Usage

```
symmetrizer(n = 1, matrix = FALSE)
```

Arguments

n	order of the symmetrizer matrix.
matrix	a logical indicating whether the symmetrizer matrix will be returned.

Details

This function is a wrapper function for the function symm.info. This function provides the information required to create the symmetrizer matrix. If option matrix = FALSE the symmetrizer matrix is stored in three vectors containing the coordinate list of indexes for rows, columns and the values.

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^2 matrix (if requested).

References

Abadir, K.M., Magnus, J.R. (2005). *Matrix Algebra*. Cambridge University Press.

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

See Also

symm.info

Examples

```
z <- symmetrizer(n = 100)
object.size(z) # 319 Kb of storage

N100 <- symmetrizer(n = 100, matrix = TRUE) # time: < 2 secs
object.size(N100) # 800 Mb of storage, do not request this matrix!

# a small example
N3 <- symmetrizer(n = 3, matrix = TRUE)
a <- matrix(rep(c(2,4,6), each = 3), ncol = 3)
a
b <- 0.5 * (a + t(a))
b
v <- N3 %*% vec(a)
all(vec(b) == as.vector(v)) # vectors are equal!
```

---

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
```

---

whitening   *Whitening transformation*

Description

Applies the whitening transformation to a data matrix based on the Cholesky decomposition of the empirical covariance matrix.

Usage

```
whitening(x, Scatter = NULL)
```

Arguments

x   vector or matrix of data with, say, p columns.

Scatter   covariance (or scatter) matrix $(p \times p)$ of the distribution, must be positive definite.
                     If NULL, the covariance matrix is estimated from the data.

Value

Returns the whitened data matrix $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}^T$, where

$$\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{S}^{-1},$$

with $\boldsymbol{S}$ the empirical covariance matrix.

References

Kessy, A., Lewin, A., Strimmer, K. (2018). Optimal whitening and decorrelation. *The American Statistician* **72**, 309-314.

Examples

```
x <- iris[,1:4]
species <- iris[,5]
pairs(x, col = species) # plot of Iris

# whitened data
z <- whitening(x)
pairs(z, col = species) # plot of
```

---

wilson.hilferty   *Wilson-Hilferty transformation*

Description

Returns the Wilson-Hilferty transformation of random variables with chi-squared distribution.

Usage

```
wilson.hilferty(x)
```

Arguments

x   vector or matrix of data with, say, p columns.

Details

Let $F = D^2/p$ be a random variable, where D^2 denotes the squared Mahalanobis distance defined as

$$D^2 = (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$$

Thus the Wilson-Hilferty transformation is given by

$$z = \frac{F^{1/3} - (1 - \frac{2}{9p})}{(\frac{2}{9p})^{1/2}}$$

and z is approximately distributed as a standard normal distribution. This is useful, for instance, in the construction of QQ-plots.

References

Wilson, E.B., and Hilferty, M.M. (1931). The distribution of chi-square. *Proceedings of the National Academy of Sciences of the United States of America* **17**, 684-688.

See Also

[cov](), [Mahalanobis]()

Examples

```
x <- iris[,1:4]
z <- wilson.hilferty(x)
par(pty = "s")
qqnorm(z, main = "Transformed distances Q-Q plot")
abline(c(0,1), col = "red", lwd = 2, lty = 2)
```

Index