Package 'fastmatrix'

January 14, 2026

Type Package

Title Fast Computation of some Matrices Useful in Statistics

Version 0.6-6

Date 2026-01-13

Maintainer Felipe Osorio <faosorios.stat@gmail.com>

Description Small set of functions designed to speed up the computation of certain matrix operations that are commonly used in statistics and econometrics. It provides efficient implementations for the computation of several structured matrices, matrix decompositions and statistical procedures, many of which have minimal memory overhead. 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://github.com/faosorios/fastmatrix

NeedsCompilation yes

LazyLoad yes

Author Felipe Osorio [aut, cre] (ORCID: <https://orcid.org/0000-0002-4675-5201>), Alonso Ogueda [aut] (ORCID: <https://orcid.org/0000-0001-5411-1484>)

Repository CRAN

Date/Publication 2026-01-14 06:10:29 UTC

Contents

`fastmatrix-package` *Fast Computation of some Matrices Useful in Statistics*

Description

Small set of functions designed to speed up the computation of certain matrix operations that are commonly used in statistics and econometrics. It provides efficient implementations for the computation of several structured matrices, matrix decompositions and statistical procedures, many of which have minimal memory overhead. Furthermore, the package provides interfaces to C code callable by another C code from other R packages.

Details

The **fastmatrix** package provides functions to the efficient construction of *duplication*, *commutation* and *symmetrizer* matrices with minimal storage requeriments. Common matrix decompositions (e.g. LU, LDL), rank-1 updates (e.g. Cholesky update), iterative solvers for linear systems and other linear algebra utilities, as well as basic matrix operations, such as Hadamard (elementwise) and Kronecker products, the Sherman-Morrison formula and the power method. The package also offers several statistical procedures, namely: the sweep operator, weighted mean and covariance (using online algorithms), ordinary least squares via multiple strategies (Cholesky, QR, SVD, sweep and conjugate gradients), ridge regression (with procedures for selecting the ridge parameter), omnibus tests for univariate normality, multivariate skewness and kurtosis measures, Mahalanobis distance (with positive-definiteness checks), Wilson-Hilferty transformation of gamma variables (useful, for instance, for goodness-of-fit of multivariate normal data), and some random number generators. Finally, the package provides interfaces for code written in C, enabling other R packages (or user-written C code) to access the C routines in the **fastmatrix** package.

Author(s)

Felipe Osorio <faosorios.stat@gmail.com>, Alonso Ogueda <aogueda@gmu.edu>

`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 $\boldsymbol{X} = (x_{tij})$ be a 3-dimensional $n \times p \times q$ where indices t, i and j indicate face, row and column, respectively. The product $\boldsymbol{Y} = \boldsymbol{AXB}$ is an $n \times r \times s$ array, with $\boldsymbol{A}$ and $\boldsymbol{B}$ are $r \times p$ and $q \times s$ matrices respectively. The elements of $\boldsymbol{Y}$ 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 $\boldsymbol{X}$ 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 $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ control points the Bezier curve is given by $B(t)$ defined as

$$B(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^k \boldsymbol{p}_k$$

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

ccc *Lin's concordance correlation coefficient*

Description

Calculates Lin's concordance correlation coefficient for evaluating the degree of agreement between measurements generated by two different methods.

Usage

```
ccc(x, data, method = "z-transform", level = 0.95, equal.means = FALSE,
    ustat = TRUE, subset, na.action)
```

Arguments

x	a formula or a numeric matrix or an object that can be coerced to a numeric matrix.
data	an optional data frame (or similar: see model.frame), used only if x is a formula. By default the variables are taken from environment(formula).
method	a character string, indicating the method for the computation of the required confidence interval. Options available are "z-transform" or "asymp" (see details in Lin, 1989, 2000).
level	the confidence level required, must be a single number between 0 and 1 (by default 95%).
equal.means	logical, should the means of the measuring devices be considered equal? In which case the restricted estimation is carried out under this assumption.
ustat	logical, should the concordance correlation coefficient be estimated using *U*-statistics?
subset	an optional expression indicating the subset of the rows of data that should be used in the fitting process.
na.action	a function that indicates what should happen when the data contain NAs.

Value

A list with class 'ccc' containing the following named components:

call	a list containing an image of the ccc call that produced the object.
x	data.frame used in the estimation process.
ccc	estimate of the concordance correlation coefficient.
var.ccc	asymptotic variance of the concordance correlation coefficient estimate.
accuracy	estimate of the accuracy (or bias) coefficient that measures how far the best-fit line deviates from a line at 45 degrees. No deviation from the 45 degree line occurs when accuracy = 1.
precision	estimate of the precision (or Pearson correlation) coefficient.

shifts	list with the location and scale shifts.
z	*Z*-transformation parameter estimate.
var.z	asymptotic variance of the *Z*-transformation parameter estimate.
confint	confidence interval for the Lin's concordance correlation coefficient.
bland	a data frame with two columns containing the average of each pair of measurements, and difference between the measurements.
center	the estimated mean vector.
cov	the estimated covariance matrix.
ustat	available only if ustat = TRUE, in which case this element corresponds to a list containing the following elements rhoc, var.rhoc, ustat, and cov.
Restricted	available only if equal.means = TRUE, in which case this element corresponds to a list containing the following elements ccc, accuracy, precision, shifts, center, and cov.

References

Bland, J., Altman, D. (1986). Statistical methods for assessing agreement between two methods of clinical measurement. *The Lancet* **327**, 307-310.

King, T.S., Chinchilli, V.M. (2001). A generalized concordance correlation coefficient for continuous and categorical data. *Statistics in Medicine* **20**, 2131-2147.

King, T.S., Chinchilli, V.M. (2001). Robust estimators of the concordance correlation coefficient. *Journal of Biopharmaceutical Statistics* **11**, 83-105.

Lin, L. (1989). A concordance correlation coefficient to evaluate reproducibility. *Biometrics* **45**, 255-268.

Lin, L. (2000). A note on the concordance correlation coefficient. *Biometrics* **56**, 324-325.

Vallejos, R., Osorio, F., Ferrer, C. (2025). A new coefficient to measure agreement between two continuous variables. doi: 10.48550/arXiv.2507.07913.

Examples

```
## data in Fig.1 from Bland and Altman (1986).
x <- list(Large = c(494,395,516,434,476,557,413,442,650,433,
          417,656,267,478,178,423,427),
          Mini  = c(512,430,520,428,500,600,364,380,658,445,
          432,626,260,477,259,350,451))
x <- as.data.frame(x)

plot(Mini ~ Large, data = x, xlim = c(100,800), ylim = c(100,800),
     xlab = "PERF by Large meter", ylab = "PERF by Mini meter")
abline(c(0,1), col = "gray", lwd = 2)

## estimating CCC
z <- ccc(~ Mini + Large, data = x, method = "asymp")
z
## output:
# Call:
```

```
# ccc(x = ~ Mini + Large, data = x, method = "asymp")
#
# Coefficients:
#  estimate  variance  accuracy precision
#   0.9427    0.0008    0.9994    0.9433
#
# Asymptotic 95% confidence interval:
#     CCC     SE  lower  upper
#  0.9427 0.0286 0.8867 0.9988
```

---

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
```

---

chi *The chi distribution*

Description

Density, distribution function, quantile function and random generation for the chi distribution with df degrees of freedom.

Usage

```
dchi(x, df = 1, log = FALSE)
pchi(q, df = 1, lower.tail = TRUE, log.p = FALSE)
qchi(p, df = 1, lower.tail = TRUE, log.p = FALSE)
rchi(n, df = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations.
df	degrees of freedom (non-negative, but can be non-integer).
log, log.p	logical; if TRUE, probabilities p are given as $\log(p)$.
lower.tail	logical; if TRUE (default), probabilities are $P(X \leq x)$, otherwise, $P(X > x)$.

Details

If df is not specified, they assume the default value of 1.

The chi distribution with ν degrees of freedom (also known as shape parameter) has density

$$f(x) = \frac{1}{2^{\nu/2-1}\Gamma(\nu/2)} x^{\nu-1} \exp(-x^2/2),$$

where $y > 0$, and $\nu > 0$.

rchi implements the routine proposed by Monahan (1987) for generating chi-distributed random variables.

Value

`dchi`, `pchi`, and `qchi` are respectively the density, distribution function and quantile function of the chi distribution. `rchi` generates random deviates drawn from the chi distribution, the length of the result is determined by `n`.

Author(s)

Felipe Osorio

References

Forbes, C., Evans, M., Hastings, N., Peacock, B. (2010). *Statistical Distributions*, 4th Ed. Wiley, New York.

Johnson, N.L., Kotz, S., Balakrishnan, N. (1994). *Continuous Univariate Distributions, Volume 1*, 2nd Ed. Wiley, New York.

Monahan, J.F. (1987). An algorithm for generating chi random variables. *ACM Transactions on Mathematical Software* **13**, 168-172.

See Also

Distributions for other standard distributions.

Examples

```
x <- rchi(1000, df = 2)
## QQ-plot for chi data against true theoretical distribution:
qqplot(qchi(ppoints(1000), df = 2), x, main = "chi QQ-plot",
  xlab = "Theoretical quantiles", ylab = "Sample quantiles")
abline(c(0,1), lwd = 2, lty = 2, col = "red")
```

`cholupdate` *Rank 1 update to Cholesky factorization*

Description

function `cholupdate` returns the upper triangular Cholesky factor of $\boldsymbol{A} + \boldsymbol{x}\boldsymbol{x}^T$, where $\boldsymbol{R}$ is the original Cholesky factor of $\boldsymbol{A} = \boldsymbol{R}^T\boldsymbol{R}$, with $\boldsymbol{x}$ 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 $\mathtt{vec}(\boldsymbol{A})$ to $\mathtt{vec}(\boldsymbol{A}^T)$.

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 dimensions of a commutation matrix $\boldsymbol{K}$ of order mn and a conformable matrix $\boldsymbol{x}$, performs one of the matrix-matrix operations:

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

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

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, which is given by

$$\boldsymbol{S}_{HD} = \frac{1}{2(n-1)} \sum_{i=2}^{n} (\boldsymbol{x}_i - \boldsymbol{x}_{i-1})(\boldsymbol{x}_i - \boldsymbol{x}_{i-1})^T.$$

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. Thus,

$$W_n = \sum_{i=1}^{n} w_i, \qquad \overline{\boldsymbol{x}}_n = \frac{1}{W_n} \sum_{i=1}^{n} w_i \boldsymbol{x}_i \qquad \boldsymbol{S}_n = \frac{1}{n} \sum_{i=1}^{n} w_i (\boldsymbol{x}_i - \overline{\boldsymbol{x}}_n)(\boldsymbol{x}_i - \overline{\boldsymbol{x}}_n)^T.$$

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 a conformable matrix $\boldsymbol{X}$, this function performs the operation: $\boldsymbol{Y} = \boldsymbol{D}_n^T \boldsymbol{X} \boldsymbol{D}_k$, where $\boldsymbol{D}_n$ and $\boldsymbol{D}_k$ 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 a conformable matrix $\boldsymbol{X}$, performs one of the matrix-matrix operations:

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

where $\boldsymbol{D}$ 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 $\boldsymbol{A}$, $\mathsf{vech}(\boldsymbol{A})$ into $\mathsf{vec}(\boldsymbol{A})$.

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 $\boldsymbol{X}$ 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)
```

floyd *Find the shortest paths in a directed graph*

Description

The Floyd-Warshall algorithm finds all shortest paths (if exist) in a directed graph.

Usage

```
floyd(x)
```

Arguments

x	adjacency matrix of a directed graph. (i, j)-entry represents the weight (cost) of the edge from i to j if one exists and ∞ otherwise.

Value

Returns a list with final costs and the shortest path between two nodes.

References

Floyd, R.W. (1962). Algorithm 97: Shortest Path. *Communications of the ACM* **5** (6), 345.

Examples

```
x <- matrix(c(0,3,Inf,5,2,0,Inf,4,Inf,1,0,Inf,Inf,Inf,2,0), nrow = 4,
            ncol = 4, byrow = TRUE)
z <- floyd(x)
z
```

---

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

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

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 $\boldsymbol{X}$ and $\boldsymbol{Y}$, 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 $\boldsymbol{X}$ (and $\boldsymbol{Y}$) 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
```

hankel *Form a symmetric Hankel matrix*

Description

Forms a symmetric Hankel matrix of order n from the values in vector $\boldsymbol{x}$ and optionally the vector $\boldsymbol{y}$.

Usage

```
hankel(x, y = NULL)
```

Arguments

x	the first column to form the Hankel matrix.
y	the last column of the Hankel matrix. If y is not provided only its first n skew diagonals are formed and the remaining elements are zeros. Otherwise, it is assumed that $x_n = y_1$, and the first entry of vector y is discarded.

Value

A symmetric Hankel matrix of order n.

Examples

```
x <- 1:4
y <- c(4,6,8,10)

# H4
hankel(x)

# H({1,2,3,4},{4,6,8,10})
hankel(x, y)
```

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)
```

---

house *Householder vector/reflection*

Description

This function returns the Householder matrix (also called Householder reflection) or the Householder vector, which are constructed based on a nonzero vector $\boldsymbol{x}$.

Usage

```
house(x, matrix = FALSE)
```

Arguments

x	an n-dimensional numeric vector.
matrix	a logical indicating whether the commutation matrix will be returned.

Details

A Householder transformation is a rank-1 modification of the identity matrix which can be used to zero out selected elements of a vector. A Householder matrix $\boldsymbol{P}$ adopts the form

$$\boldsymbol{P} - \tau \boldsymbol{u}\boldsymbol{u}^T,$$

where $\boldsymbol{u}$ is called Householder vector, and $\tau = 2/(\boldsymbol{u}^T\boldsymbol{u})$.

Value

If `matrix = TRUE` an n by n matrix is returned, otherwise `house` function return a list containing the following components:

u	the Householder vector.
tau	the scalar τ required to construct the Householder reflector.

References

Golub, G.H., van Loan, C.F. (1996). *Matrix Computations*, 3rd Edition. The Johns Hopkins University Press, Baltimore.

Examples

```
x <- c(3,1,5,1)
z <- house(x)
z

mat <- house(x, matrix = TRUE)
v <- mat %*% x
v[1,1] == minkowski(x) # TRUE
```

---

house.prod *Applying Householder matrices*

Description

This function applies the Householder matrix defined by a nonzero vector $\boldsymbol{x}$ to a matrix $\boldsymbol{A}$. Thus, the output of this function is the matrix-matrix operation:

- $\boldsymbol{Y} = \boldsymbol{P}\boldsymbol{A}$, if `side = "left"`, or
- $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{P}$, if `side = "right"`.

Usage

```
house.prod(a, x, side = "left")
```

Arguments

a	a numeric rectangular matrix.
x	an n-dimensional numeric vector.
side	a string selecting if Householder matrix is pre-multiplying a, that is side = "left" or post-multiplying a, by using side = "right".

Details

Underlying code **never** entail the explicit formation of the Householder matrix.

References

Golub, G.H., van Loan, C.F. (1996). *Matrix Computations*, 3rd Edition. The Johns Hopkins University Press, Baltimore.

Examples

```
x <- c(3,1,5,1)
u <- house(x)$u
house.prod(u, x)

y <- as.matrix(x)
house.prod(y, 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 $\boldsymbol{D}$, $\boldsymbol{L}$, and $\boldsymbol{U}$ denote the diagonal, lower triangular and upper triangular parts of a matrix $\boldsymbol{A}$. Jacobi's method solve the equation $\boldsymbol{Ax} = \boldsymbol{b}$, iteratively by rewriting $\boldsymbol{Dx} + (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x} = \boldsymbol{b}$. Assuming that $\boldsymbol{D}$ is nonsingular leads to the iteration formula

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

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
```

JarqueBera.test *Jarque-Bera test for univariate normality*

Description

Performs an omnibus test for univariate normality.

Usage

```
JarqueBera.test(x, test = "DH")
```

Arguments

x	a numeric vector containing the sample observations.
test	test statistic to be used. One of "DH" (Doornik-Hansen, the default), "JB" (Jarque-Bera), "robust" (robust modification by Gel and Gastwirth), "ALM" (Adjusted Lagrange multiplier).

Value

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

statistic	value of the statistic, i.e. the value of either Doornik-Hansen, Jarque-Bera, or Adjusted Lagrange multiplier test.
parameter	the degrees of freedom for the test statistic, which is chi-square distributed.
p.value	the p-value for the test.
skewness	the estimated skewness coefficient.
kurtosis	the estimated kurtosis coefficient.
method	a character string indicating what type of test was performed.

References

Doornik, J.A., Hansen, H. (2008). An omnibus test for univariate and multivariate normality. *Oxford Bulletin of Economics and Statistics* **70**, 927-939.

Gel, Y.R., Gastwirth, J.L. (2008). A robust modification of the Jarque-Bera test of normality. *Economics Letters* **99**, 30-32.

Jarque, C.M., Bera, A.K. (1980). Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics Letters* **6**, 255-259.

Urzua, C.M. (1996). On the correct use of omnibus tests for normality. *Economics Letters* **53**, 247-251.

Examples

```
set.seed(149)
x <- rnorm(100)
z <- JarqueBera.test(x, test = "DH")
z

set.seed(173)
x <- runif(100)
z <- JarqueBera.test(x, test = "DH")
z
```

---

kronecker.prod *Kronecker product on matrices*

Description

Computes the kronecker product of two matrices, $\boldsymbol{X}$ and $\boldsymbol{Y}$.

Usage

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

Arguments

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

Details

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

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

is called the *Kronecker product* of $\boldsymbol{X}$ and $\boldsymbol{Y}$.

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 $\boldsymbol{A}$ an n by n real matrix and an n-vector $\boldsymbol{b}$, this function constructs the Krylov matrix $\boldsymbol{K}$, where

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

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

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

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 $\boldsymbol{X} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$, where $\boldsymbol{D}$ is a diagonal matrix and $\boldsymbol{L}$ is unitary lower triangular.

The LDL decomposition of $\boldsymbol{x}$ is returned as a list with components:

`lower`	the unitary lower triangular factor $\boldsymbol{L}$.
`d`	a vector containing the diagonal elements of $\boldsymbol{D}$.

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 $\boldsymbol{Ax} = \boldsymbol{b}$ for given matrix $\boldsymbol{A}$, and vector $\boldsymbol{b}$.

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 $\boldsymbol{U}$ of the decomposition and the strict lower triangle contains information on the $\boldsymbol{L}$ 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 $\boldsymbol{X}$, the original matrix from which the lu object was constructed (because of the pivoting the $\boldsymbol{X}$ matrix is not exactly the product between $\boldsymbol{L}$ and $\boldsymbol{U}$).

extractL returns $\boldsymbol{L}$. This may be pivoted.

extractU returns $\boldsymbol{U}$.

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 $\boldsymbol{x}$ and the vector $\boldsymbol{\mu}$ = center with respect to $\boldsymbol{\Sigma}$ = cov. This is (for vector $\boldsymbol{x}$) 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)
```

`mardia.test` *Mardia test for multivariate normality*

Description

Performs Mardia's tests to assess multivariate normality based on the multivariate skewness and kurtosis coefficients.

Usage

```
mardia.test(x)
```

Arguments

`x`	matrix of data with, say, p columns.

Value

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

`skewness`	a list containig the `statistic`, the degrees of freedom (`parameter`), the `p.value` for the test and the estimated `skewness` coefficient.
`kurtosis`	a list containig the `statistic`, the `p.value` for the test and the estimated `kurtosis` coefficient.

References

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

Mardia, K.V. (1974). Applications of some measures of multivariate skewness and kurtosis in testing normality and robustness studies. *Sankhya* **36**, 115-128.

Examples

```
setosa <- iris[1:50,1:4]
z <- mardia.test(setosa)
z

set.seed(149)
Sigma <- matrix(c(10,3,3,2), ncol = 2)
x <- rmnorm(n = 300, Sigma = Sigma)
z <- mardia.test(x)
z
```

`matrix.fun` *Evaluate a matrix function*

Description

This function computes the matrix function $\boldsymbol{F} = f(\boldsymbol{A})$ where $\boldsymbol{A}$ is upper triangular by applying a Parlett recurrence.

Usage

```
matrix.fun(a, FUN = "log")
```

Arguments

`a`	an upper triangular matrix.
`FUN`	the function to be applied, by default `"log"`.

Details

The used-defined function `FUN` is evaluated at the triangular matrix argument. This function can be used in conjunction with Schur decomposition to evaluate the function of a matrix.

References

Higham, N.J. (1986). *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia.

Examples

```
a <- matrix(c(1,2,3,0,3,4,0,0,5), ncol = 3, byrow = TRUE)
fnc <- function(x) (1 + x) / x
f <- matrix.fun(a, FUN = fnc)
f

a <- matrix(c(-49,24,-64,31), ncol = 2, byrow = TRUE)
z <- schur(a)
m <- z$m
```

```
u <- z$vectors
m <- matrix.fun(m, FUN = exp)
u %*% m %*% t(u) # exp(a)
```

`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 provided, `coef` is a vector of all ones. If the exponent is zero, 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
```

`matrix.sqrt` *Matrix square root*

Description

This function computes a square root of an $n \times n$ matrix $\boldsymbol{A}$.

Usage

```
matrix.sqrt(a, method = "DB", maxiter = 50, tol = 1e-8)
```

Arguments

`a`	a square matrix.
`method`	the procedure used to obtain the square root. If `method = "DB"` (the default) the matrix square root is obtained using a Newton's method. If `method = "schur"` the Schur decomposition is considered.
`maxiter`	the maximum number of iterations. Defaults to `50`
`tol`	a numeric tolerance.

Details

A square root of a square matrix $\boldsymbol{A}$ is obtained by solving the equation $\boldsymbol{X}^2 = \boldsymbol{A}$, considering the Newton iteration proposed by Denman and Beavers (1976), or alternatively is based on the Schur decomposition.

References

Denman, E.D., Beavers, A.N. (1976). The matrix sign function and computations in systems. *Applied Mathematics and Computation* **2**, 63-94.

Higham, N.J. (1986). Newton's method for the matrix square root. *Mathematics of Computation* **46**, 537-549.

Higham, N.J. (1986). *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia.

Examples

```
a <- matrix(c(35,17,3,17,46,11,3,11,12), ncol = 3)
root <- matrix.sqrt(a) # 8 iterations

# just checking
root %*% root

root <- matrix.sqrt(a, method = "schur")
```

mchol *The modified Cholesky factorization*

Description

Compute the Cholesky factorization of a real symmetric but not necessarily positive definite matrix.

Usage

```
mchol(x)
```

Arguments

x	a symmetric but not necessarily positive definite matrix to be factored.

Value

The lower triangular factor of modified Cholesky decomposition, i.e., the matrix $\boldsymbol{L}$ such that $\boldsymbol{X} + \boldsymbol{E} = \boldsymbol{L}\boldsymbol{L}^T$, where $\boldsymbol{E}$ is a nonnegative diagonal matrix that is zero if $\boldsymbol{X}$ es sufficiently positive definite.

References

Gill, P.E., Murray, W., Wright, M.H. (1981). *Practical Optimization*. Academic Press, London.

Nocedal, J., Wright, S.J. (1999). *Numerical Optimization*. Springer, New York.

See Also

chol, ldl

Examples

```
# a non-positive-definite matrix
a <- matrix(c(4,2,1,2,6,3,1,3,-.004), ncol = 3)
try(chol(a)) # fails
z <- mchol(a)
z # triangular factor

# modified 'a' matrix
tcrossprod(z)
```

`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 variables.

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 $\boldsymbol{x}$. 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

$$b_1 = \frac{m_3}{m_2^{3/2}}, \qquad b_2 = \frac{m_4}{m_2^2}.$$

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,
  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 `NA`s. 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.
`x, y`	logicals. If `TRUE` the corresponding components of the fit (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)
```

---

rank1.update *Rank-one update*

Description

Let $\boldsymbol{A}$ be a $n \times n$ matrix and $\boldsymbol{u}$ an n-dimensional vector. Thus, the transformation

$$\boldsymbol{A} + \alpha \boldsymbol{u}\boldsymbol{u}^T,$$

is called a *rank-one update* to $\boldsymbol{A}$, where α is an scalar.

Usage

```
rank1.update(a, alpha, u, diag = FALSE)
```

Arguments

a	a numeric matrix.
alpha	a numeric scalar.
u	a numeric vector.
diag	logical. If TRUE, a is supposed to be a diagonal matrix and only its diagonal (or a vector) is required.

Details

If diag = FALSE, method of rank1.update calls BLAS level 2 subroutine DGER for computational efficiency, otherwise the special structure of the problem is used.

Value

a square matrix of the same order as a.

Examples

```
n <- 10
ones <- rep(1, n)
z <- rank1.update(0.5 * ones, 0.5, ones, diag = TRUE)
z
```

---

rball *Generation of deviates uniformly distributed in a unitary ball*

Description

Random vector generation uniformly in the unitary ball.

Usage

```
rball(n = 1, p = 2)
```

Arguments

n	the number of samples requested
p	dimension of the unitary sphere

Details

The function rball is an interface to C routines, which make calls to subroutines from BLAS.

Value

If n = 1 a p-dimensional vector, otherwise a matrix of n rows of random vectors.

References

Hormann, W., Leydold, J., Derflinger, G. (2004). *Automatic Nonuniform Random Variate Generation*. Springer, New York.

See Also

runif

Examples

```
# generate the sample
z <- rball(n = 500)

# scatterplot of a random sample of 500 points uniformly distributed
# in the unitary ball
par(pty = "s")
plot(z, xlab = "x", ylab = "y")
title("500 points in the ball", font.main = 1)
```

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, 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.
x, y	logicals. If TRUE the corresponding components of the fit (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)
```

---

rmnorm *Multivariate normal random deviates*

Description

Random number generation from the multivariate normal (Gaussian) distribution.

Usage

```
rmnorm(n = 1, mean = rep(0, nrow(Sigma)), Sigma = diag(length(mean)))
```

Arguments

n	the number of samples requested
mean	a vector giving the means of each variable
Sigma	a positive-definite covariance matrix

Details

The function rmnorm is an interface to C routines, which make calls to subroutines from LAPACK. The matrix decomposition is internally done using the Cholesky decomposition. If Sigma is not non-negative definite then there will be a warning message.

Value

If $n = 1$ a vector of the same length as mean, otherwise a matrix of n rows of random vectors.

References

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag, New York.

See Also

rnorm

Examples

```
# covariance parameters
Sigma <- matrix(c(10,3,3,2), ncol = 2)
Sigma

# generate the sample
y <- rmnorm(n = 1000, Sigma = Sigma)
var(y)

# scatterplot of a random bivariate normal sample with mean
# vector zero and covariance matrix 'Sigma'
par(pty = "s")
plot(y, xlab = "", ylab = "")
title("bivariate normal sample", font.main = 1)

# QQ-plot of transformed distances
z <- WH.normal(y)
par(pty = "s")
qqnorm(z, main = "Transformed distances QQ-plot")
abline(c(0,1), col = "red", lwd = 2, lty = 2)
```

---

rsphere *Generation of deviates uniformly located on a spherical surface*

Description

Random vector generation uniformly on the sphere.

Usage

```
rsphere(n = 1, p = 2)
```

Arguments

n	the number of samples requested
p	dimension of the unitary sphere

Details

The function rsphere is an interface to C routines, which make calls to subroutines from BLAS.

Value

If n = 1 a p-dimensional vector, otherwise a matrix of n rows of random vectors.

References

Devroye, L. (1986). *Non-Uniform Random Variate Generation.* Springer-Verlag, New York.

See Also

runif

Examples

```
# generate the sample
z <- rsphere(n = 200)

# scatterplot of a random sample of 200 points uniformly distributed
# on the unit circle
par(pty = "s")
plot(z, xlab = "x", ylab = "y")
title("200 points on the circle", font.main = 1)
```

---

scaled.condition *Scaled condition number*

Description

Compute the scaled condition number of a rectangular matrix.

Usage

```
scaled.condition(x, scales = FALSE)
```

Arguments

x	a numeric rectangular matrix.
scales	a logical value indicating whether the scaling factors that allow balancing the columns of x should be returned by the function.

Value

The columns of a rectangular matrix $\boldsymbol{X}$ are equilibrated (but not centered), then the scaled condition number is computed following the guidelines of Belsley (1990). If requested, the column scalings are returned as the attribute 'scales'.

References

Belsley, D.A. (1990). *Conditioning Diagnostics: Collinearity and Weak Data in Regression*. Wiley, New York.

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)
scaled.condition(x)
```

---

schur *The Schur decomposition of a square matrix*

Description

schur computes the Schur decomposition of an $n \times n$ real matrix $\boldsymbol{A}$.

Usage

```
schur(x, vectors = TRUE)
```

Arguments

x	a square numeric matrix to be decomposed.
vectors	logical, if TRUE (the default), then Schur vectors are returned.

Details

For an $n \times n$ real matrix $\boldsymbol{A}$, the Schur decomposition is given by,

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{M}\boldsymbol{Q}^T$$

where $\boldsymbol{Q}$ is an orthogonal matrix and $\boldsymbol{M}$ is a quasi-upper triangular matrix. The column vectors $\boldsymbol{Q}$ (if requested) are the Schur vectors of $\boldsymbol{A}$, and $\boldsymbol{M}$ is the Schur form of $\boldsymbol{A}$.

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

Value

The Schur decomposition of the matrix as computed by LAPACK. The components in the returned value correspond directly to the values returned by DGEES.

m	a matrix with the same dimensions as $\boldsymbol{X}$. The upper triangle contains the $\boldsymbol{M}$ matrix of the decomposition.
values	a vector containing the n eigenvalues of $\boldsymbol{X}$, these values **are not ordered**.
vectors	an $n \times n$ matrix whose columns contain the eigenvectors of $\boldsymbol{X}$, only available if it is requested.

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.

Examples

```
a <- matrix(c(7,12,-2,-3), ncol = 2)
z <- schur(a)
z # information of Schur decomposition

x <- matrix(c(0,0,1,2,1,0,2,2,1), ncol = 3)
z <- schur(x)
z # complex eigenvalues
```

---

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

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

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 $(\boldsymbol{A} + \boldsymbol{b}\boldsymbol{d}^T)$ where $\boldsymbol{A}$ is a $n \times n$ matrix and $\boldsymbol{b}$, $\boldsymbol{d}$ are n-dimensional vectors. Thus

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

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 matrix $\boldsymbol{N}$ of order n and a conformable matrix $\boldsymbol{X}$, performs one of the matrix-matrix operations:

- $\boldsymbol{Y} = \boldsymbol{N}\boldsymbol{X}$, if `side = "left"`, or
- $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{N}$, if `side = "right"`,

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 $\boldsymbol{X}$, that is `side = "left"` or post-multiplying $\boldsymbol{X}$, 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 $\boldsymbol{A}$, $\mathsf{vec}(\boldsymbol{A})$ into $\mathsf{vec}((\boldsymbol{A} + \boldsymbol{A}^T)/2)$.

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 $\boldsymbol{X}$.

Usage

```
vec(x)
```

Arguments

x	a numeric matrix.

Value

Let $\boldsymbol{X}$ be a n by m matrix, then vec($\boldsymbol{X}$) 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 $\boldsymbol{X}$ be a n by n matrix, then vech($\boldsymbol{X}$) is a $n(n+1)/2$-dimensional vector.

Examples

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

WH.normal *Wilson-Hilferty transformation for chi-squared variates*

Description

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

Usage

```
WH.normal(x)
```

Arguments

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

Details

Let $T = D^2/p$ 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

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

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

Mahalanobis

Examples

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

`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 Gamma distribution.

Usage

```
wilson.hilferty(x, shape, rate = 1)
```

Arguments

`x`	a numeric vector containing Gamma distributed deviates.
`shape`, `rate`	shape and rate parameters. Must be positive.

Details

Let X be a random variable following a Gamma distribution with parameters a = `shape` and b = `rate` with density

$$f(x) = \frac{b^a}{\Gamma(a)} x^{a-1} \exp(-bx),$$

where $x \geq 0$, $a > 0$, $b > 0$ and consider the random variable $T = X/(a/b)$. Thus, the Wilson-Hilferty transformation

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

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

References

Terrell, G.R. (2003). The Wilson-Hilferty transformation is locally saddlepoint. *Biometrika* **90**, 445-453.

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

`WH.normal`

Examples

```
x <- rgamma(n = 300, shape = 2, rate = 1)
z <- wilson.hilferty(x, shape = 2, rate = 1)
par(pty = "s")
qqnorm(z, main = "Transformed Gamma QQ-plot")
abline(c(0,1), col = "red", lwd = 2, lty = 2)
```

Index