

# Package ‘matlib’

October 13, 2022

**Type** Package

**Title** Matrix Functions for Teaching and Learning Linear Algebra and Multivariate Statistics

**Version** 0.9.5

**Date** 2021-08-10

**Maintainer** Michael Friendly <friendly@yorku.ca>

**Description** A collection of matrix functions for teaching and learning matrix linear algebra as used in multivariate statistical methods. These functions are mainly for tutorial purposes in learning matrix algebra ideas using R. In some cases, functions are provided for concepts available elsewhere in R, but where the function call or name is not obvious. In other cases, functions are provided to show or demonstrate an algorithm. In addition, a collection of functions are provided for drawing vector diagrams in 2D and 3D.

**License** GPL (>= 2)

**Language** en-US

**URL** <https://github.com/friendly/matlib>

**BugReports** <https://github.com/friendly/matlib/issues>

**LazyData** TRUE

**Suggests** knitr, rglwidget, rmarkdown, carData, webshot2, markdown

**Additional\_repositories** <https://dmurdoch.github.io/drat>

**Imports** xtable, MASS, rgl, car, methods

**VignetteBuilder** knitr

**RoxygenNote** 7.1.1

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Michael Friendly [aut, cre] (<<https://orcid.org/0000-0002-3237-0941>>),  
John Fox [aut],  
Phil Chalmers [aut],  
Georges Monette [ctb],  
Gaston Sanchez [ctb]

Repository CRAN

Date/Publication 2021-08-21 15:40:02 UTC

## R topics documented:

adjoint . . . . .	3
angle . . . . .	4
arc . . . . .	5
arrows3d . . . . .	6
buildTmat . . . . .	8
cholesky . . . . .	9
circle3d . . . . .	10
class . . . . .	11
cofactor . . . . .	11
cone3d . . . . .	12
corner . . . . .	13
Det . . . . .	14
echelon . . . . .	15
Eigen . . . . .	16
gaussianElimination . . . . .	17
Ginv . . . . .	18
GramSchmidt . . . . .	20
gsorth . . . . .	21
Inverse . . . . .	22
J . . . . .	23
len . . . . .	23
LU . . . . .	24
matlib . . . . .	25
matrix2latex . . . . .	27
minor . . . . .	27
MoorePenrose . . . . .	28
mpower . . . . .	29
plot.regvec3d . . . . .	30
plotEqn . . . . .	32
plotEqn3d . . . . .	34
pointOnLine . . . . .	35
powerMethod . . . . .	36
printMatEqn . . . . .	38
printMatrix . . . . .	39
Proj . . . . .	40
QR . . . . .	41
R . . . . .	43
regvec3d . . . . .	44
rowadd . . . . .	47
rowCofactors . . . . .	48
rowMinors . . . . .	49
rowmult . . . . .	50

<i>adjoint</i>	3
rowswap . . . . .	51
showEig . . . . .	51
showEqn . . . . .	52
Solve . . . . .	55
SVD . . . . .	56
svdDemo . . . . .	57
swp . . . . .	58
symMat . . . . .	60
therapy . . . . .	61
tr . . . . .	61
vandermode . . . . .	62
vec . . . . .	62
vectors . . . . .	63
vectors3d . . . . .	65
workers . . . . .	67
xprod . . . . .	68
<b>Index</b>	<b>69</b>

---

<code>adjoint</code>	<i>Calculate the Adjoint of a matrix</i>
----------------------	--

---

### Description

This function calculates the adjoint of a square matrix, defined as the transposed matrix of cofactors of all elements.

### Usage

`adjoint(A)`

### Arguments

A                    a square matrix

### Value

a matrix of the same size as A

### Author(s)

Michael Friendly

### See Also

Other determinants: [Det\(\)](#), [cofactor\(\)](#), [minor\(\)](#), [rowCofactors\(\)](#), [rowMinors\(\)](#)

**Examples**

```
A <- J(3, 3) + 2*diag(3)
adjoint(A)
```

---

angle	<i>Angle between two vectors</i>
-------	----------------------------------

---

**Description**

angle calculates the angle between two vectors.

**Usage**

```
angle(x, y, degree = TRUE)
```

**Arguments**

x	a numeric vector
y	a numeric vector
degree	logical; should the angle be computed in degrees? If FALSE the result is returned in radians

**Value**

a scalar containing the angle between the vectors

**See Also**

[len](#)

**Examples**

```
x <- c(2,1)
y <- c(1,1)
angle(x, y) # degrees
angle(x, y, degree = FALSE) # radians

# visually
xlim <- c(0,2.5)
ylim <- c(0,2)
# proper geometry requires asp=1
plot(xlim, ylim, type="n", xlab="X", ylab="Y", asp=1,
     main = expression(theta == 18.4))
abline(v=0, h=0, col="gray")
vectors(rbind(x,y), col=c("red", "blue"), cex.lab=c(2, 2))
text(.5, .37, expression(theta))
```

```
#####
x <- c(-2,1)
y <- c(1,1)
angle(x, y) # degrees
angle(x, y, degree = FALSE) # radians

# visually
xlim <- c(-2,1.5)
ylim <- c(0,2)
# proper geometry requires asp=1
plot( xlim, ylim, type="n", xlab="X", ylab="Y", asp=1,
      main = expression(theta == 108.4))
abline(v=0, h=0, col="gray")
vectors(rbind(x,y), col=c("red", "blue"), cex.lab=c(2, 2))
text(0, .4, expression(theta), cex=1.5)
```

---

arc

---

*Draw an arc showing the angle between vectors*


---

### Description

A utility function for drawing vector diagrams. Draws a circular arc to show the angle between two vectors in 2D or 3D.

### Usage

```
arc(p1, p2, p3, d = 0.1, absolute = TRUE, ...)
```

### Arguments

p1	Starting point of first vector
p2	End point of first vector, and also start of second vector
p3	End point of second vector
d	The distance from p2 along each vector for drawing their corner
absolute	logical; if TRUE, d is taken as an absolute distance along the vectors; otherwise it is calculated as a relative distance, i.e., a fraction of the length of the vectors.
...	Arguments passed to <code>link[graphics]{lines}</code> or to <code>link[rgl]{lines3d}</code>

### Details

In this implementation, the two vectors are specified by three points, p1, p2, p3, meaning a line from p1 to p2, and another line from p2 to p3.

### Value

none

## References

<https://math.stackexchange.com/questions/1507248/find-arc-between-two-tips-of-vectors-in-3d>

## See Also

Other vector diagrams: [Proj\(\)](#), [arrows3d\(\)](#), [circle3d\(\)](#), [corner\(\)](#), [plot.regvec3d\(\)](#), [pointOnLine\(\)](#), [regvec3d\(\)](#), [vectors3d\(\)](#), [vectors\(\)](#)

## Examples

```
library(rgl)
vec <- rbind(diag(3), c(1,1,1))
rownames(vec) <- c("X", "Y", "Z", "J")
open3d()
aspect3d("iso")
vectors3d(vec, col=c(rep("black",3), "red"), lwd=2)
# draw the XZ plane, whose equation is Y=0
planes3d(0, 0, 1, 0, col="gray", alpha=0.2)
# show projections of the unit vector J
segments3d(rbind( c(1,1,1), c(1, 1, 0)))
segments3d(rbind( c(0,0,0), c(1, 1, 0)))
segments3d(rbind( c(1,0,0), c(1, 1, 0)))
segments3d(rbind( c(0,1,0), c(1, 1, 0)))
segments3d(rbind( c(1,1,1), c(1, 0, 0)))

# show some orthogonal vectors
p1 <- c(0,0,0)
p2 <- c(1,1,0)
p3 <- c(1,1,1)
p4 <- c(1,0,0)
# show some angles
arc(p1, p2, p3, d=.2)
arc(p4, p1, p2, d=.2)
arc(p3, p1, p2, d=.2)
```

---

arrows3d

*Draw 3D arrows*

---

## Description

Draws nice 3D arrows with cone3ds at their tips.

## Usage

```
arrows3d(
  coords,
  headlength = 0.035,
  head = "end",
  scale = NULL,
```

```

    radius = NULL,
    ref.length = NULL,
    draw = TRUE,
    ...
)

```

### Arguments

coords	A 2n x 3 matrix giving the start and end (x,y,z) coordinates of n arrows, in pairs. The first vector in each pair is taken as the starting coordinates of the arrow, the second as the end coordinates.
headlength	Length of the arrow heads, in device units
head	Position of the arrow head. Only head="end" is presently implemented.
scale	Scale factor for base and tip of arrow head, a vector of length 3, giving relative scale factors for X, Y, Z
radius	radius of the base of the arrow head
ref.length	length of vector to be used to scale all of the arrow heads (permits drawing arrow heads of the same size as in a previous call); if NULL, arrows are scaled relative to the longest vector
draw	if TRUE (the default) draw the arrow(s)
...	rgl arguments passed down to <a href="#">segments3d</a> and <a href="#">cone3d</a> , for example, col and lwd

### Details

This function is meant to be analogous to [arrows](#), but for 3D plots using [rgl](#). headlength, scale and radius set the length, scale factor and base radius of the arrow head, a 3D cone. The units of these are all in terms of the ranges of the current rgl 3D scene.

### Value

invisibly returns the length of the vector used to scale the arrow heads

### Author(s)

January Weiner, borrowed from the [pca3d](#) package, slightly modified by John Fox

### See Also

[vectors3d](#)

Other vector diagrams: [Proj\(\)](#), [arc\(\)](#), [circle3d\(\)](#), [corner\(\)](#), [plot.regvec3d\(\)](#), [pointOnLine\(\)](#), [regvec3d\(\)](#), [vectors3d\(\)](#), [vectors\(\)](#)

### Examples

```
#none yet
```

---

 buildTmat

*Build/Get transformation matrices*


---

### Description

Recover the history of the row operations that have been performed. This function combines the transformation matrices into a single transformation matrix representing all row operations or may optionally print all the individual operations which have been performed.

### Usage

```
buildTmat(x, all = FALSE)

## S3 method for class 'trace'
as.matrix(x, ...)

## S3 method for class 'trace'
print(x, ...)
```

### Arguments

x	a matrix A, joined with a vector of constants, b, that has been passed to <a href="#">gaussianElimination</a> or the row operator matrix functions
all	logical; print individual transformation matrices?
...	additional arguments

### Value

the transformation matrix or a list of individual transformation matrices

### Author(s)

Phil Chalmers

### See Also

[echelon](#), [gaussianElimination](#)

### Examples

```
A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
b <- c(8, -11, -3)

# using row operations to reduce below diagonal to 0
Abt <- Ab <- cbind(A, b)
Abt <- rowadd(Abt, 1, 2, 3/2)
```



```
Abt <- rowadd(Abt, 1, 3, 1)
Abt <- rowadd(Abt, 2, 3, -4)
Abt

# build T matrix and multiply by original form
(T <- buildTmat(Abt))
T %*% Abt # same as Abt

# print all transformation matrices
buildTmat(Abt, TRUE)

# invert transformation matrix to reverse operations
inv(T) %*% Abt

# gaussian elimination
(soln <- gaussianElimination(A, b))
T <- buildTmat(soln)
inv(T) %*% soln
```

---

cholesky

*Cholesky Square Root of a Matrix*

---

### Description

Returns the Cholesky square root of the non-singular, symmetric matrix  $X$ . The purpose is mainly to demonstrate the algorithm used by Kennedy & Gentle (1980).

### Usage

```
cholesky(X, tol = sqrt(.Machine$double.eps))
```

### Arguments

<code>X</code>	a square symmetric matrix
<code>tol</code>	tolerance for checking for 0 pivot

### Value

the Cholesky square root of  $X$

### Author(s)

John Fox

### References

Kennedy W.J. Jr, Gentle J.E. (1980). *Statistical Computing*. Marcel Dekker.

**See Also**

[chol](#) for the base R function

[gsorth](#) for Gram-Schmidt orthogonalization of a data matrix

**Examples**

```
C <- matrix(c(1,2,3,2,5,6,3,6,10), 3, 3) # nonsingular, symmetric
C
cholesky(C)
cholesky(C) %%% t(cholesky(C)) # check
```

---

circle3d

*Draw a horizontal circle*

---

**Description**

A utility function for drawing a horizontal circle in the (x,y) plane in a 3D graph

**Usage**

```
circle3d(center, radius, segments = 100, fill = FALSE, ...)
```

**Arguments**

center	A vector of length 3.
radius	A positive number.
segments	An integer specifying the number of line segments to use to draw the circle (default, 100).
fill	logical; if TRUE, the circle is filled (the default is FALSE).
...	<b>rgl</b> material properties for the circle.

**See Also**

Other vector diagrams: [Proj\(\)](#), [arc\(\)](#), [arrows3d\(\)](#), [corner\(\)](#), [plot.regvec3d\(\)](#), [pointOnLine\(\)](#), [regvec3d\(\)](#), [vectors3d\(\)](#), [vectors\(\)](#)

**Examples**

```
ctr=c(0,0,0)
circle3d(ctr, 3, fill = TRUE)
circle3d(ctr - c(-1,-1,0), 3, col="blue")
circle3d(ctr + c(1,1,0), 3, col="red")
```

---

class	<i>Class Data Set</i>
-------	-----------------------

---

**Description**

A small artificial data set used to illustrate statistical concepts.

**Usage**

```
data("class")
```

**Format**

A data frame with 15 observations on the following 4 variables.

sex a factor with levels F M

age a numeric vector

height a numeric vector

weight a numeric vector

**Examples**

```
data(class)
plot(class)
```

---

cofactor	<i>Cofactor of A[i,j]</i>
----------	---------------------------

---

**Description**

Returns the cofactor of element (i,j) of the square matrix A, i.e., the signed minor of the sub-matrix that results when row i and column j are deleted.

**Usage**

```
cofactor(A, i, j)
```

**Arguments**

A a square matrix

i row index

j column index

**Value**

the cofactor of A[i,j]

**Author(s)**

Michael Friendly

**See Also**

[rowCofactors](#) for all cofactors of a given row

Other determinants: [Det\(\)](#), [adjoint\(\)](#), [minor\(\)](#), [rowCofactors\(\)](#), [rowMinors\(\)](#)

**Examples**

```
M <- matrix(c(4, -12, -4,
             2,  1,  3,
             -1, -3,  2), 3, 3, byrow=TRUE)
cofactor(M, 1, 1)
cofactor(M, 1, 2)
cofactor(M, 1, 3)
```

---

cone3d

*Draw a 3D cone*

---

**Description**

Draws a cone in 3D from a base point to a tip point, with a given radius at the base. This is used to draw nice arrow heads in [arrows3d](#).

**Usage**

```
cone3d(base, tip, radius = 10, col = "grey", scale = NULL, ...)
```

**Arguments**

base	coordinates of base of the cone
tip	coordinates of tip of the cone
radius	radius of the base
col	color
scale	scale factor for base and tip
...	rgl arguments passed down; see <a href="#">rgl.material</a>

**Value**

returns the integer object ID of the shape that was added to the scene

**Author(s)**

January Weiner, borrowed from from the **pca3d** package

**See Also**[arrows3d](#)**Examples**

# none yet

---

`corner`*Draw a corner showing the angle between two vectors*

---

**Description**

A utility function for drawing vector diagrams. Draws two line segments to indicate the angle between two vectors, typically used for indicating orthogonal vectors are at right angles in 2D and 3D diagrams.

**Usage**`corner(p1, p2, p3, d = 0.1, absolute = TRUE, ...)`**Arguments**

<code>p1</code>	Starting point of first vector
<code>p2</code>	End point of first vector, and also start of second vector
<code>p3</code>	End point of second vector
<code>d</code>	The distance from <code>p2</code> along each vector for drawing their corner
<code>absolute</code>	logical; if <code>TRUE</code> , <code>d</code> is taken as an absolute distance along the vectors; otherwise it is calculated as a relative distance, i.e., a fraction of the length of the vectors. See <a href="#">pointOnLine</a> for the precise definition.
<code>...</code>	Arguments passed to <code>link[graphics]{lines}</code> or to <code>link[rgl]{lines3d}</code>

**Details**

In this implementation, the two vectors are specified by three points, `p1`, `p2`, `p3`, meaning a line from `p1` to `p2`, and another line from `p2` to `p3`.

**Value**

none

**See Also**

Other vector diagrams: [Proj\(\)](#), [arc\(\)](#), [arrows3d\(\)](#), [circle3d\(\)](#), [plot.regvec3d\(\)](#), [pointOnLine\(\)](#), [regvec3d\(\)](#), [vectors3d\(\)](#), [vectors\(\)](#)

**Examples**

# none yet

---

**Det***Determinant of a Square Matrix*

---

**Description**

Returns the determinant of a square matrix  $X$ , computed either by Gaussian elimination, expansion by cofactors, or as the product of the eigenvalues of the matrix. If the latter,  $X$  must be symmetric.

**Usage**

```
Det(  
  X,  
  method = c("elimination", "eigenvalues", "cofactors"),  
  verbose = FALSE,  
  fractions = FALSE,  
  ...  
)
```

**Arguments**

<code>X</code>	a square matrix
<code>method</code>	one of "elimination" (the default), "eigenvalues", or "cofactors" (for computation by minors and cofactors)
<code>verbose</code>	logical; if TRUE, print intermediate steps
<code>fractions</code>	logical; if TRUE, try to express non-integers as rational numbers
<code>...</code>	arguments passed to <a href="#">gaussianElimination</a> or <a href="#">Eigen</a>

**Value**

the determinant of  $X$

**Author(s)**

John Fox

**See Also**

[det](#) for the base R function

[gaussianElimination](#), [Eigen](#)

Other determinants: [adjoint\(\)](#), [cofactor\(\)](#), [minor\(\)](#), [rowCofactors\(\)](#), [rowMinors\(\)](#)

**Examples**

```

A <- matrix(c(1,2,3,2,5,6,3,6,10), 3, 3) # nonsingular, symmetric
A
Det(A)
Det(A, verbose=TRUE, fractions=TRUE)
B <- matrix(1:9, 3, 3) # a singular matrix
B
Det(B)
C <- matrix(c(1, .5, .5, 1), 2, 2) # square, symmetric, nonsingular
Det(C)
Det(C, method="eigenvalues")
Det(C, method="cofactors")

```

---

echelon

*Echelon Form of a Matrix*


---

**Description**

Returns the (reduced) row-echelon form of the matrix A, using [gaussianElimination](#).

**Usage**

```
echelon(A, B, reduced = TRUE, ...)
```

**Arguments**

A	coefficient matrix
B	right-hand side vector or matrix. If B is a matrix, the result gives solutions for each column as the right-hand side of the equations with coefficients in A.
reduced	logical; should reduced row echelon form be returned? If FALSE a non-reduced row echelon form will be returned
...	other arguments passed to <a href="#">gaussianElimination</a>

**Details**

When the matrix A is square and non-singular, the reduced row-echelon result will be the identity matrix, while the row-echelon form will be an upper triangle matrix. Otherwise, the result will have some all-zero rows, and the rank of the matrix is the number of not all-zero rows.

**Value**

the reduced echelon form of X.

**Author(s)**

John Fox

**Examples**

```

A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
b <- c(8, -11, -3)
echelon(A, b, verbose=TRUE, fractions=TRUE) # reduced row-echelon form
echelon(A, b, reduced=FALSE, verbose=TRUE, fractions=TRUE) # row-echelon form

A <- matrix(c(1,2,3,4,5,6,7,8,10), 3, 3) # a nonsingular matrix
A
echelon(A, reduced=FALSE) # the row-echelon form of A
echelon(A) # the reduced row-echelon form of A

b <- 1:3
echelon(A, b) # solving the matrix equation Ax = b
echelon(A, diag(3)) # inverting A

B <- matrix(1:9, 3, 3) # a singular matrix
B
echelon(B)
echelon(B, reduced=FALSE)
echelon(B, b)
echelon(B, diag(3))

```

---

 Eigen

*Eigen Decomposition of a Square Symmetric Matrix*


---

**Description**

Eigen calculates the eigenvalues and eigenvectors of a square, symmetric matrix using the iterated QR decomposition

**Usage**

```
Eigen(X, tol = sqrt(.Machine$double.eps), max.iter = 100, retain.zeroes = TRUE)
```

**Arguments**

X	a square symmetric matrix
tol	tolerance passed to QR
max.iter	maximum number of QR iterations
retain.zeroes	logical; retain 0 eigenvalues?

**Value**

a list of two elements: values– eigenvalues, vectors– eigenvectors



**Author(s)**

John Fox and Georges Monette

**See Also**

[eigen](#)

[SVD](#)

**Examples**

```
C <- matrix(c(1,2,3,2,5,6,3,6,10), 3, 3) # nonsingular, symmetric
C
EC <- Eigen(C) # eigenanalysis of C
EC$vectors %*% diag(EC$values) %*% t(EC$vectors) # check
```

---

gaussianElimination    *Gaussian Elimination*

---

**Description**

gaussianElimination demonstrates the algorithm of row reduction used for solving systems of linear equations of the form  $Ax = B$ . Optional arguments `verbose` and `fractions` may be used to see how the algorithm works.

**Usage**

```
gaussianElimination(
  A,
  B,
  tol = sqrt(.Machine$double.eps),
  verbose = FALSE,
  latex = FALSE,
  fractions = FALSE
)

## S3 method for class 'enhancedMatrix'
print(x, ...)
```

**Arguments**

A	coefficient matrix
B	right-hand side vector or matrix. If B is a matrix, the result gives solutions for each column as the right-hand side of the equations with coefficients in A.
tol	tolerance for checking for 0 pivot
verbose	logical; if TRUE, print intermediate steps

latex	logical; if TRUE, and verbose is TRUE, print intermediate steps using LaTeX equation outputs rather than R output
fractions	logical; if TRUE, try to express non-integers as rational numbers
x	matrix to print
...	arguments to pass down

**Value**

If B is absent, returns the reduced row-echelon form of A. If B is present, returns the reduced row-echelon form of A, with the same operations applied to B.

**Author(s)**

John Fox

**Examples**

```
A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
b <- c(8, -11, -3)
gaussianElimination(A, b)
gaussianElimination(A, b, verbose=TRUE, fractions=TRUE)
gaussianElimination(A, b, verbose=TRUE, fractions=TRUE, latex=TRUE)

# determine whether matrix is solvable
gaussianElimination(A, numeric(3))

# find inverse matrix by elimination: A = I -> A^-1 A = A^-1 I -> I = A^-1
gaussianElimination(A, diag(3))
inv(A)

# works for 1-row systems (issue # 30)
A2 <- matrix(c(1, 1), nrow=1)
b2 = 2
gaussianElimination(A2, b2)
showEqn(A2, b2)
# plotEqn works for this case
plotEqn(A2, b2)
```

**Description**

Ginv returns an arbitrary generalized inverse of the matrix A, using gaussianElimination.

**Usage**

```
Ginv(A, tol = sqrt(.Machine$double.eps), verbose = FALSE, fractions = FALSE)
```

**Arguments**

A	numerical matrix
tol	tolerance for checking for 0 pivot
verbose	logical; if TRUE, print intermediate steps
fractions	logical; if TRUE, try to express non-integers as rational numbers

**Details**

A generalized inverse is a matrix  $A^-$  satisfying  $AA^-A = A$ .

The purpose of this function is mainly to show how the generalized inverse can be computed using Gaussian elimination.

**Value**

the generalized inverse of A, expressed as fractions if fractions=TRUE, or rounded

**Author(s)**

John Fox

**See Also**

[ginv](#) for a more generally usable function

**Examples**

```
A <- matrix(c(1,2,3,4,5,6,7,8,10), 3, 3) # a nonsingular matrix
A
Ginv(A, fractions=TRUE) # a generalized inverse of A = inverse of A
round(Ginv(A) %*% A, 6) # check

B <- matrix(1:9, 3, 3) # a singular matrix
B
Ginv(B, fractions=TRUE) # a generalized inverse of B
B %*% Ginv(B) %*% B # check
```

**Description**

Carries out simple Gram-Schmidt orthogonalization of a matrix. Treating the columns of the matrix  $X$  in the given order, each successive column after the first is made orthogonal to all previous columns by subtracting their projections on the current column.

**Usage**

```
GramSchmidt(  
  X,  
  normalize = TRUE,  
  verbose = FALSE,  
  tol = sqrt(.Machine$double.eps)  
)
```

**Arguments**

<code>X</code>	a matrix
<code>normalize</code>	logical; should the resulting columns be normalized to unit length?
<code>verbose</code>	logical; if TRUE, print intermediate steps
<code>tol</code>	the tolerance for detecting linear dependencies in the columns of <code>a</code> . The default is <code>.Machine\$double.eps</code>

**Value**

A matrix of the same size as  $X$ , with orthogonal columns

**Author(s)**

Phil Chalmers, John Fox

**Examples**

```
(xx <- matrix(c( 1:3, 3:1, 1, 0, -2), 3, 3))  
crossprod(xx)  
(zz <- GramSchmidt(xx, normalize=FALSE))  
zapsmall(crossprod(zz))  
  
# normalized  
(zz <- GramSchmidt(xx))  
zapsmall(crossprod(zz))  
  
# print steps  
GramSchmidt(xx, verbose=TRUE)
```

```
# A non-invertible matrix; hence, it is of deficient rank
(xx <- matrix(c( 1:3, 3:1, 1, 0, -1), 3, 3))
R(xx)
crossprod(xx)
# GramSchmidt finds an orthonormal basis
(zz <- GramSchmidt(xx))
zapsmall(crossprod(zz))
```

---

gsorth

*Gram-Schmidt Orthogonalization of a Matrix*


---

### Description

Calculates a matrix with uncorrelated columns using the Gram-Schmidt process

### Usage

```
gsorth(y, order, recenter = TRUE, rescale = TRUE, adjnames = TRUE)
```

### Arguments

y	a numeric matrix or data frame
order	if specified, a permutation of the column indices of y
recenter	logical; if TRUE, the result has same means as the original y, else means = 0 for cols 2:p
rescale	logical; if TRUE, the result has same sd as original, else, sd = residual sd
adjnames	logical; if TRUE, colnames are adjusted to Y1, Y2.1, Y3.12, ...

### Details

This function, originally from the **heplots** package has now been deprecated in **matlib**. Use [GramSchmidt](#) instead.

### Value

a matrix/data frame with uncorrelated columns

### Examples

```
## Not run:
set.seed(1234)
A <- matrix(c(1:60 + rnorm(60)), 20, 3)
cor(A)
G <- gsorth(A)
zapsmall(cor(G))

## End(Not run)
```

---

Inverse

*Inverse of a Matrix*

---

### Description

Uses `gaussianElimination` to find the inverse of a square, non-singular matrix,  $X$ .

### Usage

```
Inverse(X, tol = sqrt(.Machine$double.eps), ...)
```

### Arguments

<code>X</code>	a square numeric matrix
<code>tol</code>	tolerance for checking for 0 pivot
<code>...</code>	other arguments passed on

### Details

The method is purely didactic: The identity matrix,  $I$ , is appended to  $X$ , giving  $X|I$ . Applying Gaussian elimination gives  $I|X^{-1}$ , and the portion corresponding to  $X^{-1}$  is returned.

### Value

the inverse of  $X$

### Author(s)

John Fox

### Examples

```
A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
Inverse(A)
Inverse(A, verbose=TRUE, fractions=TRUE)
```

J

*Create a vector, matrix or array of constants***Description**

This function creates a vector, matrix or array of constants, typically used for the unit vector or unit matrix in matrix expressions.

**Usage**

```
J(..., constant = 1, dimnames = NULL)
```

**Arguments**

...	One or more arguments supplying the dimensions of the array, all non-negative integers
constant	The value of the constant used in the array
dimnames	Either NULL or the names for the dimensions.

**Details**

The "dimnames" attribute is optional: if present it is a list with one component for each dimension, either NULL or a character vector of the length given by the element of the "dim" attribute for that dimension. The list can be named, and the list names will be used as names for the dimensions.

**Examples**

```
J(3)
J(2,3)
J(2,3,2)
J(2,3, constant=2, dimnames=list(letters[1:2], LETTERS[1:3]))

X <- matrix(1:6, nrow=2, ncol=3)
dimnames(X) <- list(sex=c("M", "F"), day=c("Mon", "Wed", "Fri"))
J(2) %*% X      # column sums
X %*% J(3)     # row sums
```

len

*Length of a Vector or Column Lengths of a Matrix***Description**

len calculates the Euclidean length (also called Euclidean norm) of a vector or the length of each column of a numeric matrix.

**Usage**

```
len(X)
```

**Arguments**

`X` a numeric vector or matrix

**Value**

a scalar or vector containing the length(s)

**See Also**

[norm](#) for more general matrix norms

**Examples**

```
len(1:3)
len(matrix(1:9, 3, 3))

# distance between two vectors
len(1:3 - c(1,1,1))
```

---

LU

*LU Decomposition*


---

**Description**

LU computes the LU decomposition of a matrix,  $A$ , such that  $PA = LU$ , where  $L$  is a lower triangle matrix,  $U$  is an upper triangle, and  $P$  is a permutation matrix.

**Usage**

```
LU(A, b, tol = sqrt(.Machine$double.eps), verbose = FALSE, ...)
```

**Arguments**

`A` coefficient matrix

`b` right-hand side vector. When supplied the returned object will also contain the solved  $d$  and  $x$  elements

`tol` tolerance for checking for 0 pivot

`verbose` logical; if TRUE, print intermediate steps

`...` additional arguments passed to [showEqn](#)



**Details**

The LU decomposition is used to solve the equation  $Ax = b$  by calculating  $L(Ux - d) = 0$ , where  $Ld = b$ . If row exchanges are necessary for  $A$  then the permutation matrix  $P$  will be required to exchange the rows in  $A$ ; otherwise,  $P$  will be an identity matrix and the LU equation will be simplified to  $A = LU$ .

**Value**

A list of matrix components of the solution,  $P$ ,  $L$  and  $U$ . If  $b$  is supplied, the vectors  $d$  and  $x$  are also returned.

**Author(s)**

Phil Chalmers

**Examples**

```
A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
b <- c(8, -11, -3)
(ret <- LU(A)) # P is an identity; no row swapping
with(ret, L %*% U) # check that A = L * U
LU(A, b)

LU(A, b, verbose=TRUE)
LU(A, b, verbose=TRUE, fractions=TRUE)

# permutations required in this example
A <- matrix(c(1, 1, -1,
             2, 2, 4,
             1, -1, 1), 3, 3, byrow=TRUE)
b <- c(1, 2, 9)
(ret <- LU(A, b))
with(ret, P %*% A)
with(ret, L %*% U)
```

**Description**

These functions are designed mainly for tutorial purposes in teaching & learning matrix algebra ideas and applications to statistical methods using R.

## Details

In some cases, functions are provided for concepts available elsewhere in R, but where the function call or name is not obvious. In other cases, functions are provided to show or demonstrate an algorithm, sometimes providing a verbose argument to print the details of computations.

In addition, a collection of functions are provided for drawing vector diagrams in 2D and 3D.

These are not meant for production uses. Other methods are more efficient for larger problems.

## Topics

The functions in this package are grouped under the following topics

- Convenience functions:  
[tr](#), [R](#), [J](#), [len](#), [vec](#), [Proj](#), [mpower](#), [vandermode](#)
- Determinants: functions for calculating determinants by cofactor expansion  
[minor](#), [cofactor](#), [rowMinors](#), [rowCofactors](#)
- Elementary row operations: functions for solving linear equations "manually" by the steps used in row echelon form and Gaussian elimination  
[rowadd](#), [rowmult](#), [rowswap](#)
- Linear equations: functions to illustrate linear equations of the form  $Ax = b$   
[showEqn](#), [plotEqn](#)
- Gaussian elimination: functions for illustrating Gaussian elimination for solving systems of linear equations of the form  $Ax = b$ .  
[gaussianElimination](#), [Inverse](#), [inv](#), [echelon](#), [Ginv](#), [LU](#), [cholesky](#), [swp](#)
- Eigenvalues: functions to illustrate the algorithms for calculating eigenvalues and eigenvectors  
[eigen](#), [SVD](#), [powerMethod](#), [showEig](#)
- Vector diagrams: functions for drawing vector diagrams in 2D and 3D  
[arrows3d](#), [corner](#), [arc](#), [pointOnLine](#), [vectors](#), [vectors3d](#), [regvec3d](#)

Most of these ideas and implementations arose in courses and books by the authors. [Psychology 6140](<http://friendly.apps01.yorku.ca/psy6140/>) was a starting point. Fox (1984) introduced illustrations of vector geometry.

## References

Fox, J. Linear Statistical Models and Related Methods. John Wiley and Sons, 1984

Fox, J. and Friendly, M. (2016). "Visualizing Simultaneous Linear Equations, Geometric Vectors, and Least-Squares Regression with the matlib Package for R". *useR Conference*, Stanford, CA, June 27 - June 30, 2016.

---

matrix2latex	<i>Convert matrix to LaTeX equation</i>
--------------	---

---

**Description**

This function provides a soft-wrapper to `xtable::xtableMatharray()` with support for fractions output and square brackets.

**Usage**

```
matrix2latex(x, fractions = FALSE, brackets = TRUE, ...)
```

**Arguments**

x	a matrix
fractions	logical; if TRUE, try to express non-integers as rational numbers
brackets	logical; include square brackets around the matrices?
...	additional arguments passed to <code>xtable::xtableMatharray()</code>

**Author(s)**

Phil Chalmers

**Examples**

```
A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
b <- c(8, -11, -3)

matrix2latex(cbind(A,b))
matrix2latex(cbind(A,b), digits = 0)
matrix2latex(cbind(A/2,b), fractions = TRUE)
```

---

minor	<i>Minor of A[i,j]</i>
-------	------------------------

---

**Description**

Returns the minor of element (i,j) of the square matrix A, i.e., the determinant of the sub-matrix that results when row i and column j are deleted.

**Usage**

```
minor(A, i, j)
```

**Arguments**

A                    a square matrix  
 i                    row index  
 j                    column index

**Value**

the minor of A[i,j]

**Author(s)**

Michael Friendly

**See Also**

[rowMinors](#) for all minors of a given row  
 Other determinants: [Det\(\)](#), [adjoint\(\)](#), [cofactor\(\)](#), [rowCofactors\(\)](#), [rowMinors\(\)](#)

**Examples**

```
M <- matrix(c(4, -12, -4,
              2,  1,  3,
              -1, -3,  2), 3, 3, byrow=TRUE)
minor(M, 1, 1)
minor(M, 1, 2)
minor(M, 1, 3)
```

---

MoorePenrose

*Moore-Penrose inverse of a matrix*

---

**Description**

The Moore-Penrose inverse is a generalization of the regular inverse of a square, non-singular, symmetric matrix to other cases (rectangular, singular), yet retain similar properties to a regular inverse.

**Usage**

```
MoorePenrose(X, tol = sqrt(.Machine$double.eps))
```

**Arguments**

X                    A numeric matrix  
 tol                  Tolerance for a singular (rank-deficient) matrix

**Value**

The Moore-Penrose inverse of X

**Examples**

```

X <- matrix(rnorm(20), ncol=2)
# introduce a linear dependency in X[,3]
X <- cbind(X, 1.5*X[, 1] - pi*X[, 2])

Y <- MoorePenrose(X)
# demonstrate some properties of the M-P inverse
# X Y X = X
round(X %**% Y %**% X - X, 8)
# Y X Y = Y
round(Y %**% X %**% Y - Y, 8)
# X Y = t(X Y)
round(X %**% Y - t(X %**% Y), 8)
# Y X = t(Y X)
round(Y %**% X - t(Y %**% X), 8)

```

---

mpower

*Matrix Power*


---

**Description**

A simple function to demonstrate calculating the power of a square symmetric matrix in terms of its eigenvalues and eigenvectors.

**Usage**

```
mpower(A, p, tol = sqrt(.Machine$double.eps))
```

**Arguments**

A	a square symmetric matrix
p	matrix power, not necessarily a positive integer
tol	tolerance for determining if the matrix is symmetric

**Details**

The matrix power  $p$  can be a fraction or other non-integer. For example,  $p=1/2$  and  $p=1/3$  give a square-root and cube-root of the matrix.

Negative powers are also allowed. For example,  $p=-1$  gives the inverse and  $p=-1/2$  gives the inverse square-root.

**Value**

A raised to the power  $p$ :  $A^p$

**See Also**

The `{%**%}` operator in the **expm** package is far more efficient

**Examples**

```
C <- matrix(c(1,2,3,2,5,6,3,6,10), 3, 3) # nonsingular, symmetric
C
mpower(C, 2)
zapsmall(mpower(C, -1))
solve(C) # check
```

---

plot.regvec3d

*Plot method for regvec3d objects*


---

**Description**

The plot method for regvec3d objects uses the low-level graphics tools in this package to draw 3D and 3D vector diagrams reflecting the partial and marginal relations of  $y$  to  $x_1$  and  $x_2$  in a bivariate multiple linear regression model,  $\text{lm}(y \sim x_1 + x_2)$ .

The summary method prints the vectors and their vector lengths, followed by the summary for the model.

**Usage**

```
## S3 method for class 'regvec3d'
plot(
  x,
  y,
  dimension = 3,
  col = c("black", "red", "blue", "brown", "lightgray"),
  col.plane = "gray",
  cex.lab = 1.2,
  show.base = 2,
  show.marginal = FALSE,
  show.hplane = TRUE,
  show.angles = TRUE,
  error.sphere = c("none", "e", "y.hat"),
  scale.error.sphere = x$scale,
  level.error.sphere = 0.95,
  grid = FALSE,
  add = FALSE,
  ...
)

## S3 method for class 'regvec3d'
summary(object, ...)

## S3 method for class 'regvec3d'
print(x, ...)
```

**Arguments**

x	A “regvec3d” object
y	Ignored; only included for compatibility with the S3 generic
dimension	Number of dimensions to plot: 3 (default) or 2
col	A vector of 5 colors. col[1] is used for the y and residual (e) vectors, and for x1 and x2; col[2] is used for the vectors y -> yhat and y -> e; col[3] is used for the vectors yhat -> b1 and yhat -> b2;
col.plane	Color of the base plane in a 3D plot or axes in a 2D plot
cex.lab	character expansion applied to vector labels. May be a number or numeric vector corresponding to the the rows of X, recycled as necessary.
show.base	If show.base > 0, draws the base plane in a 3D plot; if show.base > 1, the plane is drawn thicker
show.marginal	If TRUE also draws lines showing the marginal relations of y on x1 and on x2
show.hplane	If TRUE, draws the plane defined by y, yhat and the origin in the 3D
show.angles	If TRUE, draw and label the angle between the x1 and x2 and between y and yhat, corresponding respectively to the correlation between the xs and the multiple correlation
error.sphere	Plot a sphere (or in 2D, a circle) of radius proportional to the length of the residual vector, centered either at the origin (“e”) or at the fitted-values vector (“y.hat”; the default is “none”).
scale.error.sphere	Whether to scale the error sphere if error.sphere=“y.hat”; defaults to TRUE if the vectors representing the variables are scaled, in which case the oblique projections of the error spheres can represent confidence intervals for the coefficients; otherwise defaults to FALSE.
level.error.sphere	The confidence level for the error sphere, applied if scale.error.sphere=TRUE.
grid	If TRUE, draws a light grid on the base plane
add	If TRUE, add to the current plot; otherwise start a new rgl or plot window
...	Parameters passed down to functions [unused now]
object	A regvec3d object for the summary method

**Details**

A 3D diagram shows the vector y and the plane formed by the predictors, x1 and x2, where all variables are represented in deviation form, so that the intercept need not be included.

A 2D diagram, using the first two columns of the result, can be used to show the projection of the space in the x1, x2 plane.

The drawing functions `vectors` and `link{vectors3d}` used by the `plot.regvec3d` method only work reasonably well if the variables are shown on commensurate scales, i.e., with either `scale=TRUE` or `normalize=TRUE`.

**Value**

None

**References**

Fox, J. (2016). *Applied Regression Analysis and Generalized Linear Models*, 3rd ed., Sage, Chapter 10.

**See Also**

[regvec3d](#), [vectors3d](#), [vectors](#)

Other vector diagrams: [Proj\(\)](#), [arc\(\)](#), [arrows3d\(\)](#), [circle3d\(\)](#), [corner\(\)](#), [pointOnLine\(\)](#), [regvec3d\(\)](#), [vectors3d\(\)](#), [vectors\(\)](#)

**Examples**

```
if (require(carData)) {
  data("Duncan", package="carData")
  dunc.reg <- regvec3d(prestige ~ income + education, data=Duncan)
  plot(dunc.reg)
  plot(dunc.reg, dimension=2)
  plot(dunc.reg, error.sphere="e")
  summary(dunc.reg)

  # Example showing Simpson's paradox
  data("States", package="carData")
  states.vec <- regvec3d(SATM ~ pay + percent, data=States, scale=TRUE)
  plot(states.vec, show.marginal=TRUE)
  plot(states.vec, show.marginal=TRUE, dimension=2)
  summary(states.vec)
}
```

---

 plotEqn

*Plot Linear Equations*


---

**Description**

Shows what matrices  $A$ ,  $b$  look like as the system of linear equations,  $Ax = b$  with two unknowns,  $x_1$ ,  $x_2$ , by plotting a line for each equation.

**Usage**

```
plotEqn(
  A,
  b,
  vars,
  xlim,
  ylim,
```



```

col = 1:nrow(A),
lwd = 2,
lty = 1,
axes = TRUE,
labels = TRUE,
solution = TRUE
)

```

### Arguments

A	either the matrix of coefficients of a system of linear equations, or the matrix <code>cbind(A,b)</code> . The A matrix must have two columns.
b	if supplied, the vector of constants on the right hand side of the equations, of length matching the number of rows of A.
vars	a numeric or character vector of names of the variables. If supplied, the length must be equal to the number of unknowns in the equations, i.e., 2. The default is <code>c(expression(x[1]), expression(x[2]))</code> .
xlim	horizontal axis limits for the first variable
ylim	vertical axis limits for the second variable; if missing, ylim is calculated from the range of the set of equations over the xlim.
col	scalar or vector of colors for the lines, recycled as necessary
lwd	scalar or vector of line widths for the lines, recycled as necessary
lty	scalar or vector of line types for the lines, recycled as necessary
axes	logical; draw horizontal and vertical axes through (0,0)?
labels	logical, or a vector of character labels for the equations; if TRUE, each equation is labeled using the character string resulting from <code>showEqn</code> , modified so that the xs are properly subscripted.
solution	logical; should the solution points for pairs of equations be marked?

### Value

nothing; used for the side effect of making a plot

### Author(s)

Michael Friendly

### References

Fox, J. and Friendly, M. (2016). "Visualizing Simultaneous Linear Equations, Geometric Vectors, and Least-Squares Regression with the `matlib` Package for R". *useR Conference*, Stanford, CA, June 27 - June 30, 2016.

### See Also

[showEqn](#)

**Examples**

```
# consistent equations
A<- matrix(c(1,2,3, -1, 2, 1),3,2)
b <- c(2,1,3)
showEqn(A, b)
plotEqn(A,b)

# inconsistent equations
b <- c(2,1,6)
showEqn(A, b)
plotEqn(A,b)
```

---

plotEqn3d

*Plot Linear Equations in 3D*


---

**Description**

Shows what matrices  $A$ ,  $b$  look like as the system of linear equations,  $Ax = b$  with three unknowns,  $x_1$ ,  $x_2$ , and  $x_3$ , by plotting a plane for each equation.

**Usage**

```
plotEqn3d(
  A,
  b,
  vars,
  xlim = c(-2, 2),
  ylim = c(-2, 2),
  zlim,
  col = 2:(nrow(A) + 1),
  alpha = 0.9,
  labels = FALSE,
  solution = TRUE,
  axes = TRUE,
  lit = FALSE
)
```

**Arguments**

<b>A</b>	either the matrix of coefficients of a system of linear equations, or the matrix <code>cbind(A,b)</code> The A matrix must have three columns.
<b>b</b>	if supplied, the vector of constants on the right hand side of the equations, of length matching the number of rows of A.
<b>vars</b>	a numeric or character vector of names of the variables. If supplied, the length must be equal to the number of unknowns in the equations. The default is <code>paste0("x", 1:ncol(A))</code> .
<b>xlim</b>	axis limits for the first variable

ylim	axis limits for the second variable
zlim	horizontal axis limits for the second variable; if missing, zlim is calculated from the range of the set of equations over the xlim and ylim
col	scalar or vector of colors for the lines, recycled as necessary
alpha	transparency applied to each plane
labels	logical, or a vector of character labels for the equations; not yet implemented.
solution	logical; should the solution point for all equations be marked (if possible)
axes	logical; whether to frame the plot with coordinate axes
lit	logical, specifying if lighting calculation should take place on geometry; see <a href="#">rgl.material</a>

**Value**

nothing; used for the side effect of making a plot

**Author(s)**

Michael Friendly, John Fox

**References**

Fox, J. and Friendly, M. (2016). "Visualizing Simultaneous Linear Equations, Geometric Vectors, and Least-Squares Regression with the matlib Package for R". *useR Conference*, Stanford, CA, June 27 - June 30, 2016.

**Examples**

```
# three consistent equations in three unknowns
A <- matrix(c(13, -4, 2, -4, 11, -2, 2, -2, 8), 3,3)
b <- c(1,2,4)
plotEqn3d(A,b)
```

---

pointOnLine

*Position of a point along a line*

---

**Description**

A utility function for drawing vector diagrams. Find position of an interpolated point along a line from x1 to x2.

**Usage**

```
pointOnLine(x1, x2, d, absolute = TRUE)
```

**Arguments**

x1	A vector of length 2 or 3, representing the starting point of a line in 2D or 3D space
x2	A vector of length 2 or 3, representing the ending point of a line in 2D or 3D space
d	The distance along the line from x1 to x2 of the point to be found.
absolute	logical; if TRUE, d is taken as an absolute distance along the line; otherwise it is calculated as a relative distance, i.e., a fraction of the length of the line.

**Details**

The function takes a step of length d along the line defined by the difference between the two points,  $x2 - x1$ . When `absolute=FALSE`, this step is proportional to the difference, while when `absolute=TRUE`, the difference is first scaled to unit length so that the step is always of length d. Note that the physical length of a line in different directions in a graph depends on the aspect ratio of the plot axes, and lines of the same length will only appear equal if the aspect ratio is one (`asp=1` in 2D, or `aspect3d("iso")` in 3D).

**Value**

The interpolated point, a vector of the same length as x1

**See Also**

Other vector diagrams: [Proj\(\)](#), [arc\(\)](#), [arrows3d\(\)](#), [circle3d\(\)](#), [corner\(\)](#), [plot.regvec3d\(\)](#), [regvec3d\(\)](#), [vectors3d\(\)](#), [vectors\(\)](#)

**Examples**

```
x1 <- c(0, 0)
x2 <- c(1, 4)
pointOnLine(x1, x2, 0.5)
pointOnLine(x1, x2, 0.5, absolute=FALSE)
pointOnLine(x1, x2, 1.1)

y1 <- c(1, 2, 3)
y2 <- c(3, 2, 1)
pointOnLine(y1, y2, 0.5)
pointOnLine(y1, y2, 0.5, absolute=FALSE)
```

---

powerMethod

*Power Method for Eigenvectors*

---

**Description**

Finds a dominant eigenvalue,  $\lambda_1$ , and its corresponding eigenvector,  $v_1$ , of a square matrix by applying Hotelling's (1933) Power Method with scaling.

**Usage**

```
powerMethod(A, v = NULL, eps = 1e-06, maxiter = 100, plot = FALSE)
```

**Arguments**

A	a square numeric matrix
v	optional starting vector; if not supplied, it uses a unit vector of length equal to the number of rows / columns of x.
eps	convergence threshold for terminating iterations
maxiter	maximum number of iterations
plot	logical; if TRUE, plot the series of iterated eigenvectors?

**Details**

The method is based upon the fact that repeated multiplication of a matrix  $A$  by a trial vector  $v_1^{(k)}$  converges to the value of the eigenvector,

$$v_1^{(k+1)} = Av_1^{(k)} / \|Av_1^{(k)}\|$$

The corresponding eigenvalue is then found as

$$\lambda_1 = \frac{v_1^T Av_1}{v_1^T v_1}$$

In pre-computer days, this method could be extended to find subsequent eigenvalue - eigenvector pairs by "deflation", i.e., by applying the method again to the new matrix.  $A - \lambda_1 v_1 v_1^T$ .

This method is still used in some computer-intensive applications with huge matrices where only the dominant eigenvector is required, e.g., the Google Page Rank algorithm.

**Value**

a list containing the eigenvector (vector), eigenvalue (value), iterations (iter), and iteration history (vector\_iterations)

**Author(s)**

Gaston Sanchez (from matrixkit)

**References**

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24, 417-441, and 498-520.

**Examples**

```

A <- cbind(c(7, 3), c(3, 6))
powerMethod(A)
eigen(A)$values[1] # check
eigen(A)$vectors[,1]

# demonstrate how the power method converges to a solution
powerMethod(A, v = c(-.5, 1), plot = TRUE)

B <- cbind(c(1, 2, 0), c(2, 1, 3), c(0, 3, 1))
(rv <- powerMethod(B))

# deflate to find 2nd latent vector
l <- rv$value
v <- c(rv$vector)
B1 <- B - l * outer(v, v)
powerMethod(B1)
eigen(B)$vectors # check

# a positive, semi-definite matrix, with eigenvalues 12, 6, 0
C <- matrix(c(7, 4, 1, 4, 4, 4, 1, 4, 7), 3, 3)
eigen(C)$vectors
powerMethod(C)

```

---

printMatEqn

---

*Print Matrices or Matrix Operations Side by Side*


---

**Description**

This function is designed to print a collection of matrices, vectors, character strings and matrix expressions side by side. A typical use is to illustrate matrix equations in a compact and comprehensible way.

**Usage**

```
printMatEqn(..., space = 1, tol = sqrt(.Machine$double.eps), fractions = FALSE)
```

**Arguments**

...	matrices and character operations to be passed and printed to the console. These can include named arguments, character string operation symbols (e.g., "+")
space	amount of blank spaces to place around operations such as "+", "-", "=", etc
tol	tolerance for rounding
fractions	logical; if TRUE, try to express non-integers as rational numbers

**Value**

NULL; A formatted sequence of matrices and matrix operations is printed to the console

**Author(s)**

Phil Chalmers

**See Also**[showEqn](#)**Examples**

```

A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
x <- c(2, 3, -1)

# provide implicit or explicit labels
printMatEqn(AA = A, "*" , xx = x, '=' , b = A %*% x)
printMatEqn(A, "*" , x, '=' , b = A %*% x)
printMatEqn(A, "*" , x, '=' , A %*% x)

# compare with showEqn
b <- c(4, 2, 1)
printMatEqn(A, x=paste0("x", 1:3), "=", b)
showEqn(A, b)

# decimal example
A <- matrix(c(0.5, 1, 3, 0.75, 2.8, 4), nrow = 2)
x <- c(0.5, 3.7, 2.3)
y <- c(0.7, -1.2)
b <- A %*% x - y

printMatEqn(A, "*" , x, "-" , y, "=", b)
printMatEqn(A, "*" , x, "-" , y, "=", b, fractions=TRUE)

```

---

printMatrix

---

*Print a matrix, allowing fractions or LaTeX output*


---

**Description**

Print a matrix, allowing fractions or LaTeX output

**Usage**

```

printMatrix(
  A,
  parent = TRUE,
  fractions = FALSE,
  latex = FALSE,

```

```

    tol = sqrt(.Machine$double.eps)
  )

```

### Arguments

A	A numeric matrix
parent	flag used to search in the parent envir for suitable definitions of other arguments. Set to TRUE (the default) if you want to only use the inputs provided.
fractions	If TRUE, print numbers as rational fractions
latex	If TRUE, print the matrix in LaTeX format
tol	Tolerance for rounding small numbers to 0

### Value

The formatted matrix

### See Also

[fractions](#)

### Examples

```

A <- matrix(1:12, 3, 4) / 6
printMatrix(A, fractions=TRUE)
printMatrix(A, latex=TRUE)

```

---

Proj

*Projection of Vector y on columns of X*

---

### Description

Fitting a linear model,  $\text{lm}(y \sim X)$ , by least squares can be thought of geometrically as the orthogonal projection of  $y$  on the column space of  $X$ . This function is designed to allow exploration of projections and orthogonality.

### Usage

```
Proj(y, X, list = FALSE)
```

### Arguments

y	a vector, treated as a one-column matrix
X	a vector or matrix. Number of rows of y and X must match
list	logical; if FALSE, return just the projected vector; otherwise returns a list



**Details**

The projection is defined as  $Py$  where  $P = X(X'X)^{-1}X'$  and  $X^{-}$  is a generalized inverse.

**Value**

the projection of  $y$  on  $X$  (if `list=FALSE`) or a list with elements  $y$  and  $P$

**Author(s)**

Michael Friendly

**See Also**

Other vector diagrams: [arc\(\)](#), [arrows3d\(\)](#), [circle3d\(\)](#), [corner\(\)](#), [plot.regvec3d\(\)](#), [pointOnLine\(\)](#), [regvec3d\(\)](#), [vectors3d\(\)](#), [vectors\(\)](#)

**Examples**

```
X <- matrix( c(1, 1, 1, 1, 1, -1, 1, -1), 4,2, byrow=TRUE)
y <- 1:4
Proj(y, X[,1]) # project y on unit vector
Proj(y, X[,2])
Proj(y, X)

# project unit vector on line between two points
y <- c(1,1)
p1 <- c(0,0)
p2 <- c(1,0)
Proj(y, cbind(p1, p2))

# orthogonal complements
y <- 1:4
yp <- Proj(y, X, list=TRUE)
yp$y
P <- yp$P
IP <- diag(4) - P
yc <- c(IP %*% y)
crossprod(yp$y, yc)

# P is idempotent: P P = P
P %*% P
all.equal(P, P %*% P)
```

**Description**

QR computes the QR decomposition of a matrix,  $X$ , that is an orthonormal matrix,  $Q$  and an upper triangular matrix,  $R$ , such that  $X = QR$ .

**Usage**

```
QR(X, tol = sqrt(.Machine$double.eps))
```

**Arguments**

X	a numeric matrix
tol	tolerance for detecting linear dependencies in the columns of X

**Details**

The QR decomposition plays an important role in many statistical techniques. In particular it can be used to solve the equation  $Ax = b$  for given matrix  $A$  and vector  $b$ . The function is included here simply to show the algorithm of Gram-Schmidt orthogonalization. The standard `qr` function is faster and more accurate.

**Value**

a list of three elements, consisting of an orthonormal matrix Q, an upper triangular matrix R, and the rank of the matrix X

**Author(s)**

John Fox and Georges Monette

**See Also**

[qr](#)

**Examples**

```
A <- matrix(c(1,2,3,4,5,6,7,8,10), 3, 3) # a square nonsingular matrix
res <- QR(A)
res
q <- res$Q
zapsmall( t(q) %*% q) # check that q' q = I
r <- res$R
q %*% r # check that q r = A

# relation to determinant: det(A) = prod(diag(R))
det(A)
prod(diag(r))

B <- matrix(1:9, 3, 3) # a singular matrix
QR(B)
```

---

R

*Rank of a Matrix*

---

### Description

Returns the rank of a matrix  $X$ , using the QR decomposition, [QR](#). Included here as a simple function, because rank does something different and it is not obvious what to use for matrix rank.

### Usage

```
R(X)
```

### Arguments

$X$                     a matrix

### Value

rank of  $X$

### See Also

[qr](#)

### Examples

```
M <- outer(1:3, 3:1)
M
R(M)

M <- matrix(1:9, 3, 3)
M
R(M)
# why rank=2?
echelon(M)

set.seed(1234)
M <- matrix(sample(1:9), 3, 3)
M
R(M)
```

regvec3d

*Vector space representation of a two-variable regression model***Description**

regvec3d calculates the 3D vectors that represent the projection of a two-variable multiple regression model from n-D *observation* space into the 3D mean-deviation *variable* space that they span, thus showing the regression of y on x1 and x2 in the model  $lm(y \sim x1 + x2)$ . The result can be used to draw 2D and 3D vector diagrams accurately reflecting the partial and marginal relations of y to x1 and x2 as vectors in this representation.

**Usage**

```
regvec3d(x1, ...)
```

```
## S3 method for class 'formula'
regvec3d(
  formula,
  data = NULL,
  which = 1:2,
  name.x1,
  name.x2,
  name.y,
  name.e,
  name.y.hat,
  name.b1.x1,
  name.b2.x2,
  abbreviate = 0,
  ...
)
```

```
## Default S3 method:
regvec3d(
  x1,
  x2,
  y,
  scale = FALSE,
  normalize = TRUE,
  name.x1 = deparse(substitute(x1)),
  name.x2 = deparse(substitute(x2)),
  name.y = deparse(substitute(y)),
  name.e = "residuals",
  name.y.hat = paste0(name.y, "hat"),
  name.b1.x1 = paste0("b1", name.x1),
  name.b2.x2 = paste0("b2", name.x2),
  name.y1.hat = paste0(name.y, "hat 1"),
  name.y2.hat = paste0(name.y, "hat 2"),
```

```
    ...
  )
```

### Arguments

x1	The generic argument or the first predictor passed to the default method
...	Arguments passed to methods
formula	A two-sided formula for the linear regression model. It must contain two quantitative predictors (x1 and x2) on the right-hand-side. If further predictors are included, y, x1 and x2 are taken as residuals from their linear fits on these variables.
data	A data frame in which the variables in the model are found
which	Indices of predictor variables in the model taken as x1 and x2
name.x1	Name for x1 to be used in the result and plots. By default, this is taken as the name of the x1 variable in the formula, possibly abbreviated according to <code>abbreviate</code> .
name.x2	Ditto for the name of x2
name.y	Ditto for the name of y
name.e	Name for the residual vector. Default: "residuals"
name.y.hat	Name for the fitted vector
name.b1.x1	Name for the vector corresponding to the partial coefficient of x1
name.b2.x2	Name for the vector corresponding to the partial coefficient of x2
abbreviate	An integer. If <code>abbreviate &gt; 0</code> , the names of x1, x2 and y are abbreviated to this length before being combined with the other <code>name.*</code> arguments
x2	second predictor variable in the model
y	response variable in the model
scale	logical; if TRUE, standardize each of y, x1, x2 to standard scores
normalize	logical; if TRUE, normalize each vector relative to the maximum length of all
name.y1.hat	Name for the vector corresponding to the marginal coefficient of x1
name.y2.hat	Name for the vector corresponding to the marginal coefficient of x2

### Details

If additional variables are included in the model, e.g.,  $lm(y \sim x1 + x2 + x3 + \dots)$ , then y, x1 and x2 are all taken as *residuals* from their separate linear fits on  $x3 + \dots$ , thus showing their partial relations net of (or adjusting for) these additional predictors.

A 3D diagram shows the vector y and the plane formed by the predictors, x1 and x2, where all variables are represented in deviation form, so that the intercept need not be included.

A 2D diagram, using the first two columns of the result, can be used to show the projection of the space in the x1, x2 plane.

In these views, the ANOVA representation of the various sums of squares for the regression predictors appears as the lengths of the various vectors. For example, the error sum of squares is the

squared length of the  $e$  vector, and the regression sum of squares is the squared length of the  $\hat{y}$  vector.

The drawing functions `vectors` and `link{vectors3d}` used by the `plot.regvec3d` method only work reasonably well if the variables are shown on commensurate scales, i.e., with either `scale=TRUE` or `normalize=TRUE`.

## Value

An object of class “regvec3d”, containing the following components

<code>model</code>	The “lm” object corresponding to $\text{lm}(y \sim x1 + x2)$ .
<code>vectors</code>	A $9 \times 3$ matrix, whose rows correspond to the variables in the model, the residual vector, the fitted vector, the partial fits for $x1$ , $x2$ , and the marginal fits of $y$ on $x1$ and $x2$ . The columns effectively represent $x1$ , $x2$ , and $y$ , but are named “x”, “y” and “z”.

## Methods (by class)

- `formula`: Formula method for `regvec3d`
- `default`: Default method for `regvec3d`

## References

Fox, J. (2016). *Applied Regression Analysis and Generalized Linear Models*, 3rd ed., Sage, Chapter 10.

Fox, J. and Friendly, M. (2016). "Visualizing Simultaneous Linear Equations, Geometric Vectors, and Least-Squares Regression with the `matlib` Package for R". *useR Conference*, Stanford, CA, June 27 - June 30, 2016.

## See Also

[plot.regvec3d](#)

Other vector diagrams: [Proj\(\)](#), [arc\(\)](#), [arrows3d\(\)](#), [circle3d\(\)](#), [corner\(\)](#), [plot.regvec3d\(\)](#), [pointOnLine\(\)](#), [vectors3d\(\)](#), [vectors\(\)](#)

## Examples

```
library(rgl)
therapy.vec <- regvec3d(therapy ~ perstest + IE, data=therapy)
therapy.vec
plot(therapy.vec, col.plane="darkgreen")
plot(therapy.vec, dimension=2)
```

---

rowadd	<i>Add multiples of rows to other rows</i>
--------	--

---

### Description

The elementary row operation `rowadd` adds multiples of one or more rows to other rows of a matrix. This is usually used as a means to solve systems of linear equations, of the form  $Ax = b$ , and `rowadd` corresponds to adding equals to equals.

### Usage

```
rowadd(x, from, to, mult)
```

### Arguments

<code>x</code>	a numeric matrix, possibly consisting of the coefficient matrix, $A$ , joined with a vector of constants, $b$ .
<code>from</code>	the index of one or more source rows. If <code>from</code> is a vector, it must have the same length as <code>to</code> .
<code>to</code>	the index of one or more destination rows
<code>mult</code>	the multiplier(s)

### Details

The functions `rowmult` and `rowswap` complete the basic operations used in reduction to row echelon form and Gaussian elimination. These functions are used for demonstration purposes.

### Value

the matrix `x`, as modified

### See Also

[echelon](#), [gaussianElimination](#)

Other elementary row operations: `rowmult()`, `rowswap()`

### Examples

```
A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
b <- c(8, -11, -3)

# using row operations to reduce below diagonal to 0
Ab <- cbind(A, b)
(Ab <- rowadd(Ab, 1, 2, 3/2)) # row 2 <- row 2 + 3/2 row 1
(Ab <- rowadd(Ab, 1, 3, 1))  # row 3 <- row 3 + 1 row 1
```

```
(Ab <- rowadd(Ab, 2, 3, -4)) # row 3 <- row 3 - 4 row 2
# multiply to make diagonals = 1
(Ab <- rowmult(Ab, 1:3, c(1/2, 2, -1)))
# The matrix is now in triangular form

# Could continue to reduce above diagonal to zero
echelon(A, b, verbose=TRUE, fractions=TRUE)
```

---

rowCofactors

*Row Cofactors of A[i,]*


---

### Description

Returns the vector of cofactors of row  $i$  of the square matrix  $A$ . The determinant,  $\text{Det}(A)$ , can then be found as  $M[i,] \%*\% \text{rowCofactors}(M, i)$  for any row,  $i$ .

### Usage

```
rowCofactors(A, i)
```

### Arguments

$A$	a square matrix
$i$	row index

### Value

a vector of the cofactors of  $A[i,]$

### Author(s)

Michael Friendly

### See Also

[Det](#) for the determinant

Other determinants: [Det\(\)](#), [adjoint\(\)](#), [cofactor\(\)](#), [minor\(\)](#), [rowMinors\(\)](#)

### Examples

```
M <- matrix(c(4, -12, -4,
              2,  1,  3,
              -1, -3,  2), 3, 3, byrow=TRUE)
minor(M, 1, 1)
minor(M, 1, 2)
minor(M, 1, 3)
rowCofactors(M, 1)
Det(M)
```



```
# expansion by cofactors of row 1
M[1,] %*% rowCofactors(M,1)
```

---

rowMinors

*Row Minors of A[i,]*

---

### Description

Returns the vector of minors of row  $i$  of the square matrix  $A$

### Usage

```
rowMinors(A, i)
```

### Arguments

$A$	a square matrix
$i$	row index

### Value

a vector of the minors of  $A[i,]$

### Author(s)

Michael Friendly

### See Also

Other determinants: [Det\(\)](#), [adjoint\(\)](#), [cofactor\(\)](#), [minor\(\)](#), [rowCofactors\(\)](#)

### Examples

```
M <- matrix(c(4, -12, -4,
              2,  1,  3,
              -1, -3,  2), 3, 3, byrow=TRUE)
minor(M, 1, 1)
minor(M, 1, 2)
minor(M, 1, 3)
rowMinors(M, 1)
```

---

`rowmult`*Multiply Rows by Constants*

---

**Description**

Multiplies one or more rows of a matrix by constants. This corresponds to multiplying or dividing equations by constants.

**Usage**

```
rowmult(x, row, mult)
```

**Arguments**

<code>x</code>	a matrix, possibly consisting of the coefficient matrix, <code>A</code> , joined with a vector of constants, <code>b</code> .
<code>row</code>	index of one or more rows.
<code>mult</code>	row multiplier(s)

**Value**

the matrix `x`, modified

**See Also**

[echelon](#), [gaussianElimination](#)

Other elementary row operations: [rowadd\(\)](#), [rowswap\(\)](#)

**Examples**

```
A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
b <- c(8, -11, -3)

# using row operations to reduce below diagonal to 0
Ab <- cbind(A, b)
(Ab <- rowadd(Ab, 1, 2, 3/2)) # row 2 <- row 2 + 3/2 row 1
(Ab <- rowadd(Ab, 1, 3, 1))  # row 3 <- row 3 + 1 row 1
(Ab <- rowadd(Ab, 2, 3, -4))
# multiply to make diagonals = 1
(Ab <- rowmult(Ab, 1:3, c(1/2, 2, -1)))
# The matrix is now in triangular form
```

---

rowswap	<i>Interchange two rows of a matrix</i>
---------	---

---

**Description**

This elementary row operation corresponds to interchanging two equations.

**Usage**

```
rowswap(x, from, to)
```

**Arguments**

x	a matrix, possibly consisting of the coefficient matrix, A, joined with a vector of constants, b.
from	source row.
to	destination row

**Value**

the matrix x, with rows from and to interchanged

**See Also**

[echelon](#), [gaussianElimination](#)

Other elementary row operations: [rowadd\(\)](#), [rowmult\(\)](#)

---

showEig	<i>Show the eigenvectors associated with a covariance matrix</i>
---------	--

---

**Description**

This function is designed for illustrating the eigenvectors associated with the covariance matrix for a given bivariate data set. It draws a data ellipse of the data and adds vectors showing the eigenvectors of the covariance matrix.

**Usage**

```
showEig(
  X,
  col.vec = "blue",
  lwd.vec = 3,
  mult = sqrt(qchisq(levels, 2)),
  asp = 1,
  levels = c(0.5, 0.95),
```

```

    plot.points = TRUE,
    add = !plot.points,
    ...
  )

```

### Arguments

<code>X</code>	A two-column matrix or data frame
<code>col.vec</code>	color for eigenvectors
<code>lwd.vec</code>	line width for eigenvectors
<code>mult</code>	length multiplier(s) for eigenvectors
<code>asp</code>	aspect ratio of plot, set to <code>asp=1</code> by default, and passed to <code>dataEllipse</code>
<code>levels</code>	passed to <code>dataEllipse</code> determining the coverage of the data ellipse(s)
<code>plot.points</code>	logical; should the points be plotted?
<code>add</code>	logical; should this call add to an existing plot?
<code>...</code>	other arguments passed to <code>link[car]{dataEllipse}</code>

### Author(s)

Michael Friendly

### See Also

[dataEllipse](#)

### Examples

```

x <- rnorm(200)
y <- .5 * x + .5 * rnorm(200)
X <- cbind(x,y)
showEig(X)

# Duncan data
data(Duncan, package="carData")
showEig(Duncan[, 2:3], levels=0.68)
showEig(Duncan[,2:3], levels=0.68, robust=TRUE, add=TRUE, fill=TRUE)

```

---

showEqn

*Show Matrices (A, b) as Linear Equations*

---

### Description

Shows what matrices  $A, b$  look like as the system of linear equations,  $Ax = b$ , but written out as a set of equations.

**Usage**

```
showEqn(
  A,
  b,
  vars,
  simplify = FALSE,
  reduce = FALSE,
  fractions = FALSE,
  latex = FALSE
)
```

**Arguments**

A	either the matrix of coefficients of a system of linear equations, or the matrix <code>cbind(A,b)</code> . Alternatively, can be of class 'lm' to print the equations for the design matrix in a linear regression model
b	if supplied, the vector of constants on the right hand side of the equations. When omitted the values <code>b1</code> , <code>b2</code> , ..., <code>bn</code> will be used as placeholders
vars	a numeric or character vector of names of the variables. If supplied, the length must be equal to the number of unknowns in the equations. The default is <code>paste0("x", 1:ncol(A))</code> .
simplify	logical; try to simplify the equations?
reduce	logical; only show the unique linear equations
fractions	logical; express numbers as rational fractions?
latex	logical; print equations in a form suitable for LaTeX output?

**Value**

a one-column character matrix, one row for each equation

**Author(s)**

Michael Friendly, John Fox, and Phil Chalmers

**References**

Fox, J. and Friendly, M. (2016). "Visualizing Simultaneous Linear Equations, Geometric Vectors, and Least-Squares Regression with the `matlib` Package for R". *useR Conference*, Stanford, CA, June 27 - June 30, 2016.

**See Also**

[plotEqn](#), [plotEqn3d](#)

**Examples**

```

A <- matrix(c(2, 1, -1,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
b <- c(8, -11, -3)
showEqn(A, b)
# show numerically
x <- solve(A, b)
showEqn(A, b, vars=x)

showEqn(A, b, simplify=TRUE)
showEqn(A, b, latex=TRUE)

# lower triangle of equation with zeros omitted (for back solving)
A <- matrix(c(2, 1, 2,
             -3, -1, 2,
             -2, 1, 2), 3, 3, byrow=TRUE)
U <- LU(A)$U
showEqn(U, simplify=TRUE, fractions=TRUE)
showEqn(U, b, simplify=TRUE, fractions=TRUE)

#####
# Linear models Design Matrices
data(mtcars)
ancova <- lm(mpg ~ wt + vs, mtcars)
summary(ancova)
showEqn(ancova)
showEqn(ancova, simplify=TRUE)
showEqn(ancova, vars=round(coef(ancova),2))
showEqn(ancova, vars=round(coef(ancova),2), simplify=TRUE)

twoway_int <- lm(mpg ~ vs * am, mtcars)
summary(twoway_int)
car::Anova(twoway_int)
showEqn(twoway_int)
showEqn(twoway_int, reduce=TRUE)
showEqn(twoway_int, reduce=TRUE, simplify=TRUE)

# Piece-wise linear regression
x <- c(1:10, 13:22)
y <- numeric(20)
y[1:10] <- 20:11 + rnorm(10, 0, 1.5)
y[11:20] <- seq(11, 15, len=10) + rnorm(10, 0, 1.5)
plot(x, y, pch = 16)

x2 <- as.numeric(x > 10)
mod <- lm(y ~ x + I((x - 10) * x2))
summary(mod)
lines(x, fitted(mod))
showEqn(mod)
showEqn(mod, vars=round(coef(mod),2))
showEqn(mod, simplify=TRUE)

```

---

Solve	<i>Solve and Display Solutions for Systems of Linear Simultaneous Equations</i>
-------	---

---

**Description**

Solve the equation system  $Ax = b$ , given the coefficient matrix  $A$  and right-hand side vector  $b$ , using `link{gaussianElimination}`. Display the solutions using `showEqn`.

**Usage**

```
Solve(
  A,
  b = rep(0, nrow(A)),
  vars,
  verbose = FALSE,
  simplify = TRUE,
  fractions = FALSE,
  ...
)
```

**Arguments**

A,	the matrix of coefficients of a system of linear equations
b,	the vector of constants on the right hand side of the equations. The default is a vector of zeros, giving the homogeneous equations $Ax = 0$ .
vars	a numeric or character vector of names of the variables. If supplied, the length must be equal to the number of unknowns in the equations. The default is <code>paste0("x", 1:ncol(A))</code> .
verbose,	logical; show the steps of the Gaussian elimination algorithm?
simplify	logical; try to simplify the equations?
fractions	logical; express numbers as rational fractions?
...,	arguments to be passed to <code>link{gaussianElimination}</code> and <code>showEqn</code>

**Details**

This function mimics the base function `solve` when supplied with two arguments,  $(A, b)$ , but gives a prettier result, as a set of equations for the solution. The call `solve(A)` with a single argument overloads this, returning the inverse of the matrix  $A$ . For that sense, use the function `inv` instead.

**Value**

the function is used primarily for its side effect of printing the solution in a readable form, but it invisibly returns the solution as a character vector

**Author(s)**

John Fox

**See Also**[gaussianElimination](#), [showEqn](#) [inv](#), [solve](#)**Examples**

```

A1 <- matrix(c(2, 1, -1,
              -3, -1, 2,
              -2, 1, 2), 3, 3, byrow=TRUE)
b1 <- c(8, -11, -3)
Solve(A1, b1) # unique solution

A2 <- matrix(1:9, 3, 3)
b2 <- 1:3
Solve(A2, b2, fractions=TRUE) # underdetermined

b3 <- c(1, 2, 4)
Solve(A2, b3, fractions=TRUE) # overdetermined

```

SVD

*Singular Value Decomposition of a Matrix***Description**

Compute the singular-value decomposition of a matrix  $X$  either by Jacobi rotations (the default) or from the eigenstructure of  $X'X$  using [Eigen](#). Both methods are iterative. The result consists of two orthonormal matrices,  $U$ , and  $V$  and the vector  $d$  of singular values, such that  $X = U \text{diag}(d) V'$ .

**Usage**

```

SVD(
  X,
  method = c("Jacobi", "eigen"),
  tol = sqrt(.Machine$double.eps),
  max.iter = 100
)

```

**Arguments**

<code>X</code>	a square symmetric matrix
<code>method</code>	either "Jacobi" (the default) or "eigen"
<code>tol</code>	zero and convergence tolerance
<code>max.iter</code>	maximum number of iterations



**Details**

The default method is more numerically stable, but the eigenstructure method is much simpler. Singular values of zero are not retained in the solution.

**Value**

a list of three elements: d– singular values, U– left singular vectors, V– right singular vectors

**Author(s)**

John Fox and Georges Monette

**See Also**

[svd](#), the standard svd function

[Eigen](#)

**Examples**

```
C <- matrix(c(1,2,3,2,5,6,3,6,10), 3, 3) # nonsingular, symmetric
C
SVD(C)

# least squares by the SVD
data("workers")
X <- cbind(1, as.matrix(workers[, c("Experience", "Skill")]))
head(X)
y <- workers$Income
head(y)
(svd <- SVD(X))
VdU <- svd$V %%% diag(1/svd$d) %%%t(svd$U)
(b <- VdU %%% y)
coef(lm(Income ~ Experience + Skill, data=workers))
```

---

svdDemo

*Demonstrate the SVD for a 3 x 3 matrix*


---

**Description**

This function draws an rgl scene consisting of a representation of the identity matrix and a 3 x 3 matrix A, together with the corresponding representation of the matrices U, D, and V in the SVD decomposition,  $A = U D V'$ .

**Usage**

```
svdDemo(A, shape = c("cube", "sphere"), alpha = 0.7, col = rainbow(6))
```

**Arguments**

A	A 3 x 3 numeric matrix
shape	Basic shape used to represent the identity matrix: "cube" or "sphere"
alpha	transparency value used to draw the shape
col	Vector of 6 colors for the faces of the basic cube

**Value**

Nothing

**Author(s)**

Original idea from Duncan Murdoch

**Examples**

```
A <- matrix(c(1,2,0.1, 0.1,1,0.1, 0.1,0.1,0.5), 3,3)
svdDemo(A)

## Not run:
B <- matrix(c( 1, 0, 1, 0, 2, 0, 1, 0, 2), 3, 3)
svdDemo(B)

# a positive, semi-definite matrix with eigenvalues 12, 6, 0
C <- matrix(c(7, 4, 1, 4, 4, 4, 1, 4, 7), 3, 3)
svdDemo(C)

## End(Not run)
```

---

swp

*The Matrix Sweep Operator*

---

**Description**

The swp function “sweeps” a matrix on the rows and columns given in index to produce a new matrix with those rows and columns “partialled out” by orthogonalization. This was defined as a fundamental statistical operation in multivariate methods by Beaton (1964) and expanded by Dempster (1969). It is closely related to orthogonal projection, but applied to a cross-products or covariance matrix, rather than to data.

**Usage**

```
swp(M, index)
```

**Arguments**

M	a numeric matrix
index	a numeric vector indicating the rows/columns to be swept. The entries must be less than or equal to the number of rows or columns in M. If missing, the function sweeps on all rows/columns $1:\min(\dim(M))$ .

**Details**

If M is the partitioned matrix

$$\begin{bmatrix} \mathbf{R} & \mathbf{S} \\ \mathbf{T} & \mathbf{U} \end{bmatrix}$$

where  $R$  is  $q \times q$  then  $\text{swp}(M, 1:q)$  gives

$$\begin{bmatrix} \mathbf{R}^{-1} & \mathbf{R}^{-1}\mathbf{S} \\ -\mathbf{TR}^{-1} & \mathbf{U} - \mathbf{TR}^{-1}\mathbf{S} \end{bmatrix}$$

**Value**

the matrix M with rows and columns in indices swept.

**References**

Beaton, A. E. (1964), *The Use of Special Matrix Operations in Statistical Calculus*, Princeton, NJ: Educational Testing Service.

Dempster, A. P. (1969) *Elements of Continuous Multivariate Analysis*. Addison-Wesley Publ. Co., Reading, Mass.

**See Also**

[Proj, QR](#)

**Examples**

```
data(therapy)
mod3 <- lm(therapy ~ perstest + IE + sex, data=therapy)
X <- model.matrix(mod3)
XY <- cbind(X, therapy=therapy$therapy)
XY
M <- crossprod(XY)
swp(M, 1)
swp(M, 1:2)
```

---

symMat	<i>Create a Symmetric Matrix from a Vector</i>
--------	--

---

### Description

Creates a square symmetric matrix from a vector.

### Usage

```
symMat(x, diag = TRUE, byrow = FALSE, names = FALSE)
```

### Arguments

x	A numeric vector used to fill the upper or lower triangle of the matrix.
diag	Logical. If TRUE (the default), the diagonals of the created matrix are replaced by elements of x; otherwise, the diagonals of the created matrix are replaced by "1".
byrow	Logical. If FALSE (the default), the created matrix is filled by columns; otherwise, the matrix is filled by rows.
names	Either a logical or a character vector of names for the rows and columns of the matrix. If FALSE, no names are assigned; if TRUE, rows and columns are named X1, X2, ... .

### Value

A symmetric square matrix based on column major ordering of the elements in x.

### Author(s)

Originally from metaSEM: :vec2symMat, Mike W.-L. Cheung <mikewlcheung@nus.edu.sg>; modified by Michael Friendly

### Examples

```
symMat(1:6)
symMat(1:6, byrow=TRUE)
symMat(5:0, diag=FALSE)
```

---

therapy

*Therapy Data*

---

### Description

A toy data set on outcome in therapy in relation to a personality test (perstest) and a scale of internal-external locus of control (IE) used to illustrate linear and multiple regression.

### Usage

```
data("therapy")
```

### Format

A data frame with 10 observations on the following 4 variables.

sex a factor with levels F M

perstest score on a personality test, a numeric vector

therapy outcome in psychotherapy, a numeric vector

IE score on a scale of internal-external locus of control, a numeric vector

### Examples

```
data(therapy)
plot(therapy ~ perstest, data=therapy, pch=16)
abline(lm(therapy ~ perstest, data=therapy), col="red")

plot(therapy ~ perstest, data=therapy, cex=1.5, pch=16,
     col=ifelse(sex=="M", "red", "blue"))
```

---

tr

*Trace of a Matrix*

---

### Description

Calculates the trace of a square numeric matrix, i.e., the sum of its diagonal elements

### Usage

```
tr(X)
```

### Arguments

X a numeric matrix

**Value**

a numeric value, the sum of `diag(X)`

**Examples**

```
X <- matrix(1:9, 3, 3)
tr(X)
```

---

vandermode

*Vandermode Matrix*


---

**Description**

The function returns the Vandermode matrix of a numeric vector, `x`, whose columns are the vector raised to the powers  $0:n$ .

**Usage**

```
vandermode(x, n)
```

**Arguments**

<code>x</code>	a numeric vector
<code>n</code>	a numeric scalar

**Value**

a matrix of size `length(x) x n`

**Examples**

```
vandermode(1:5, 4)
```

---

vec

*Vectorize a Matrix*


---

**Description**

Returns a 1-column matrix, stacking the columns of `x`, a matrix or vector. Also supports comma-separated inputs similar to the concatenation function `c`.

**Usage**

```
vec(x, ...)
```

**Arguments**

`x`                    A matrix or vector  
`...`                    (optional) additional objects to be stacked

**Value**

A one-column matrix containing the elements of `x` and `...` in column order

**Examples**

```
vec(1:3)
vec(matrix(1:6, 2, 3))
vec(c("hello", "world"))
vec("hello", "world")
vec(1:3, "hello", "world")
```

---

vectors

---

*Draw geometric vectors in 2D*


---

**Description**

This function draws vectors in a 2D plot, in a way that facilitates constructing vector diagrams. It allows vectors to be specified as rows of a matrix, and can draw labels on the vectors.

**Usage**

```
vectors(
  X,
  origin = c(0, 0),
  lwd = 2,
  angle = 13,
  length = 0.15,
  labels = TRUE,
  cex.lab = 1.5,
  pos.lab = 4,
  frac.lab = 1,
  ...
)
```

**Arguments**

`X`                    a vector or two-column matrix representing a set of geometric vectors; if a matrix, one vector is drawn for each row

`origin`                the origin from which they are drawn, a vector of length 2.

`lwd`                    line width(s) for the vectors, a constant or vector of length equal to the number of rows of `X`.

angle	the angle argument passed to <a href="#">arrows</a> determining the angle of arrow heads.
length	the length argument passed to <a href="#">arrows</a> determining the length of arrow heads.
labels	a logical or a character vector of labels for the vectors. If TRUE and X is a matrix, labels are taken from rownames(X). If NULL, no labels are drawn.
cex.lab	character expansion applied to vector labels. May be a number or numeric vector corresponding to the the rows of X, recycled as necessary.
pos.lab	label position relative to the label point as in <a href="#">text</a> , recycled as necessary.
frac.lab	location of label point, as a fraction of the distance between origin and X, recycled as necessary. Values frac.lab > 1 locate the label beyond the end of the vector.
...	other arguments passed on to graphics functions.

### Value

none

### See Also

[arrows](#), [text](#)

Other vector diagrams: [Proj\(\)](#), [arc\(\)](#), [arrows3d\(\)](#), [circle3d\(\)](#), [corner\(\)](#), [plot.regvec3d\(\)](#), [pointOnLine\(\)](#), [regvec3d\(\)](#), [vectors3d\(\)](#)

### Examples

```
# shows addition of vectors
u <- c(3,1)
v <- c(1,3)
sum <- u+v

xlim <- c(0,5)
ylim <- c(0,5)
# proper geometry requires asp=1
plot( xlim, ylim, type="n", xlab="X", ylab="Y", asp=1)
abline(v=0, h=0, col="gray")

vectors(rbind(u,v,`u+v`=sum), col=c("red", "blue", "purple"), cex.lab=c(2, 2, 2.2))
# show the opposing sides of the parallelogram
vectors(sum, origin=u, col="red", lty=2)
vectors(sum, origin=v, col="blue", lty=2)

# projection of vectors
vectors(Proj(v,u), labels="P(v,u)", lwd=3)
vectors(v, origin=Proj(v,u))
corner(c(0,0), Proj(v,u), v, col="grey")
```



vectors3d

*Draw 3D vectors***Description**

This function draws vectors in a 3D plot, in a way that facilitates constructing vector diagrams. It allows vectors to be specified as rows of a matrix, and can draw labels on the vectors.

**Usage**

```
vectors3d(
  X,
  origin = c(0, 0, 0),
  headlength = 0.035,
  ref.length = NULL,
  radius = 1/60,
  labels = TRUE,
  cex.lab = 1.2,
  adj.lab = 0.5,
  frac.lab = 1.1,
  draw = TRUE,
  ...
)
```

**Arguments**

<code>X</code>	a vector or three-column matrix representing a set of geometric vectors; if a matrix, one vector is drawn for each row
<code>origin</code>	the origin from which they are drawn, a vector of length 3.
<code>headlength</code>	the headlength argument passed to <a href="#">arrows3d</a> determining the length of arrow heads
<code>ref.length</code>	vector length to be used in scaling arrow heads so that they are all the same size; if NULL the longest vector is used to scale the arrow heads
<code>radius</code>	radius of the base of the arrow heads
<code>labels</code>	a logical or a character vector of labels for the vectors. If TRUE and X is a matrix, labels are taken from <code>rownames(X)</code> . If FALSE or NULL, no labels are drawn.
<code>cex.lab</code>	character expansion applied to vector labels. May be a number or numeric vector corresponding to the the rows of X, recycled as necessary.
<code>adj.lab</code>	label position relative to the label point as in <a href="#">text3d</a> , recycled as necessary.
<code>frac.lab</code>	location of label point, as a fraction of the distance between <code>origin</code> and X, recycled as necessary. Values <code>frac.lab &gt; 1</code> locate the label beyond the end of the vector.
<code>draw</code>	if TRUE (the default), draw the vector(s).
<code>...</code>	other arguments passed on to graphics functions.

**Value**

invisibly returns the vector `ref.length` used to scale arrow heads

**Bugs**

At present, the color (`color=`) argument is not handled as expected when more than one vector is to be drawn.

**Author(s)**

Michael Friendly

**See Also**

[arrows3d](#), [codetexts3d](#), [codergl.material](#)

Other vector diagrams: [Proj\(\)](#), [arc\(\)](#), [arrows3d\(\)](#), [circle3d\(\)](#), [corner\(\)](#), [plot.regvec3d\(\)](#), [pointOnLine\(\)](#), [regvec3d\(\)](#), [vectors\(\)](#)

**Examples**

```
vec <- rbind(diag(3), c(1,1,1))
rownames(vec) <- c("X", "Y", "Z", "J")
library(rgl)
open3d()
vectors3d(vec, color=c(rep("black",3), "red"), lwd=2)
# draw the XZ plane, whose equation is Y=0
planes3d(0, 0, 1, 0, col="gray", alpha=0.2)
vectors3d(c(1,1,0), col="green", lwd=2)
# show projections of the unit vector J
segments3d(rbind(c(1,1,1), c(1, 1, 0)))
segments3d(rbind(c(0,0,0), c(1, 1, 0)))
segments3d(rbind(c(1,0,0), c(1, 1, 0)))
segments3d(rbind(c(0,1,0), c(1, 1, 0)))
# show some orthogonal vectors
p1 <- c(0,0,0)
p2 <- c(1,1,0)
p3 <- c(1,1,1)
p4 <- c(1,0,0)
corner(p1, p2, p3, col="red")
corner(p1, p4, p2, col="red")
corner(p1, p4, p3, col="blue")

rgl.bringtotop()
```

---

workers

*Workers Data*

---

### Description

A toy data set comprised of information on workers Income in relation to other variables, used for illustrating linear and multiple regression.

### Usage

```
data("workers")
```

### Format

A data frame with 10 observations on the following 4 variables.

Income income from the job, a numeric vector

Experience number of years of experience, a numeric vector

Skill skill level in the job, a numeric vector

Gender a factor with levels Female Male

### Examples

```
data(workers)
plot(Income ~ Experience, data=workers, main="Income ~ Experience", pch=20, cex=2)

# simple linear regression
reg1 <- lm(Income ~ Experience, data=workers)
abline(reg1, col="red", lwd=3)

# quadratic fit?
plot(Income ~ Experience, data=workers, main="Income ~ poly(Experience,2)", pch=20, cex=2)
reg2 <- lm(Income ~ poly(Experience,2), data=workers)
fit2 <- predict(reg2)
abline(reg1, col="red", lwd=1, lty=1)
lines(workers$Experience, fit2, col="blue", lwd=3)

# How does Income depend on a factor?
plot(Income ~ Gender, data=workers, main="Income ~ Gender")
points(workers$Gender, jitter(workers$Income), cex=2, pch=20)
means<-aggregate(workers$Income,list(workers$Gender),mean)
points(means,col="red", pch="+", cex=2)
lines(means,col="red", lwd=2)
```

xprod

*Generalized Vector Cross Product***Description**

Given two linearly independent length 3 vectors **a** and **b**, the cross product,  $\mathbf{a} \times \mathbf{b}$  (read "a cross b"), is a vector that is perpendicular to both **a** and **b** thus normal to the plane containing them.

**Usage**

```
xprod(...)
```

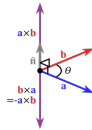
**Arguments**

... N-1 linearly independent vectors of the same length, N.

**Details**

A generalization of this idea applies to two or more dimensional vectors.

See: [[https://en.wikipedia.org/wiki/Cross\\_product](https://en.wikipedia.org/wiki/Cross_product)] for geometric and algebraic properties.

**Value**

Returns the generalized vector cross-product, a vector of length N.

**Author(s)**

Matthew Lundberg, in a [Stack Overflow post][<https://stackoverflow.com/questions/36798301/r-compute-cross-product-of-vectors-physics>]

**Examples**

```
xprod(1:3, 4:6)

# This works for an dimension
xprod(c(0,1))           # 2d
xprod(c(1,0,0), c(0,1,0)) # 3d
xprod(c(1,1,1), c(0,1,0)) # 3d
xprod(c(1,0,0,0), c(0,1,0,0), c(0,0,1,0)) # 4d
```

# Index

- \* **datasets**
  - class, 11
  - therapy, 61
  - workers, 67
- \* **determinants**
  - adjoint, 3
  - cofactor, 11
  - Det, 14
  - minor, 27
  - rowCofactors, 48
  - rowMinors, 49
- \* **elementary row operations**
  - rowadd, 47
  - rowmult, 50
  - rowswap, 51
- \* **matrix of elementary row operations**
  - buildTmat, 8
- \* **vector diagrams**
  - arc, 5
  - arrows3d, 6
  - circle3d, 10
  - corner, 13
  - plot.regvec3d, 30
  - pointOnLine, 35
  - Proj, 40
  - regvec3d, 44
  - vectors, 63
  - vectors3d, 65
- adjoint, 3, 12, 14, 28, 48, 49
- angle, 4
- arc, 5, 7, 10, 13, 26, 32, 36, 41, 46, 64, 66
- arrows, 7, 64
- arrows3d, 6, 6, 10, 12, 13, 26, 32, 36, 41, 46, 64–66
- as.matrix.trace (buildTmat), 8
- buildTmat, 8
- c, 62
- chol, 10
- cholesky, 9, 26
- circle3d, 6, 7, 10, 13, 32, 36, 41, 46, 64, 66
- class, 11
- cofactor, 3, 11, 14, 26, 28, 48, 49
- cone3d, 12
- corner, 6, 7, 10, 13, 26, 32, 36, 41, 46, 64, 66
- dataEllipse, 52
- Det, 3, 12, 14, 28, 48, 49
- det, 14
- echelon, 8, 15, 26, 47, 50, 51
- Eigen, 14, 16, 56, 57
- eigen, 17, 26
- fractions, 40
- gaussianElimination, 8, 14, 15, 17, 22, 26, 47, 50, 51, 56
- Ginv, 18, 26
- ginv, 19
- GramSchmidt, 20, 21
- gsorth, 10, 21
- inv, 26, 55, 56
- inv (Inverse), 22
- Inverse, 22, 26
- J, 23, 26
- len, 4, 23, 26
- LU, 24, 26
- matlib, 25
- matlib-package (matlib), 25
- matrix2latex, 27
- minor, 3, 12, 14, 26, 27, 48, 49
- MoorePenrose, 28
- mpower, 26, 29
- norm, 24

plot.regvec3d, [6](#), [7](#), [10](#), [13](#), [30](#), [31](#), [36](#), [41](#),  
[46](#), [64](#), [66](#)  
plotEqn, [26](#), [32](#), [53](#)  
plotEqn3d, [34](#), [53](#)  
pointOnLine, [6](#), [7](#), [10](#), [13](#), [26](#), [32](#), [35](#), [41](#), [46](#),  
[64](#), [66](#)  
powerMethod, [26](#), [36](#)  
print.enhancedMatrix  
    (gaussianElimination), [17](#)  
print.regvec3d (plot.regvec3d), [30](#)  
print.trace (buildTmat), [8](#)  
printMatEqn, [38](#)  
printMatrix, [39](#)  
Proj, [6](#), [7](#), [10](#), [13](#), [26](#), [32](#), [36](#), [40](#), [46](#), [59](#), [64](#), [66](#)  
  
QR, [16](#), [41](#), [43](#), [59](#)  
qr, [42](#), [43](#)  
  
R, [26](#), [43](#)  
regvec3d, [6](#), [7](#), [10](#), [13](#), [26](#), [32](#), [36](#), [41](#), [44](#), [64](#),  
[66](#)  
rgl, [7](#)  
rgl.material, [12](#), [35](#), [66](#)  
rowadd, [26](#), [47](#), [50](#), [51](#)  
rowCofactors, [3](#), [12](#), [14](#), [26](#), [28](#), [48](#), [49](#)  
rowMinors, [3](#), [12](#), [14](#), [26](#), [28](#), [48](#), [49](#)  
rowmult, [26](#), [47](#), [50](#), [51](#)  
rowswap, [26](#), [47](#), [50](#), [51](#)  
  
segments3d, [7](#)  
showEig, [26](#), [51](#)  
showEqn, [24](#), [26](#), [33](#), [39](#), [52](#), [55](#), [56](#)  
Solve, [55](#)  
solve, [55](#), [56](#)  
summary.regvec3d (plot.regvec3d), [30](#)  
SVD, [17](#), [26](#), [56](#)  
svd, [57](#)  
svdDemo, [57](#)  
swp, [26](#), [58](#)  
symMat, [60](#)  
  
text, [64](#)  
text3d, [65](#)  
texts3d, [66](#)  
therapy, [61](#)  
tr, [26](#), [61](#)  
  
vandermode, [26](#), [62](#)  
vec, [26](#), [62](#)  
  
vectors, [6](#), [7](#), [10](#), [13](#), [26](#), [31](#), [32](#), [36](#), [41](#), [46](#),  
[63](#), [66](#)  
vectors3d, [6](#), [7](#), [10](#), [13](#), [26](#), [32](#), [36](#), [41](#), [46](#), [64](#),  
[65](#)  
workers, [67](#)  
xprod, [68](#)