

# Package ‘lintools’

July 22, 2025

**Maintainer** Mark van der Loo <mark.vanderloo@gmail.com>

**License** GPL-3

**Title** Manipulation of Linear Systems of (in)Equalities

**Type** Package

**LazyLoad** yes

**Description** Variable elimination (Gaussian elimination, Fourier-Motzkin elimination), Moore-Penrose pseudoinverse, reduction to reduced row echelon form, value substitution, projecting a vector on the convex polytope described by a system of (in)equations, simplify systems by removing spurious columns and rows and collapse implied equalities, test if a matrix is totally unimodular, compute variable ranges implied by linear (in)equalities.

**Version** 0.1.7

**URL** <https://github.com/data-cleaning/lintools>

**BugReports** <https://github.com/data-cleaning/lintools/issues>

**Imports** utils

**Suggests** tinytest, knitr, rmarkdown

**VignetteBuilder** knitr

**RoxygenNote** 7.2.1

**NeedsCompilation** yes

**Author** Mark van der Loo [aut, cre],  
Edwin de Jonge [aut]

**Repository** CRAN

**Date/Publication** 2023-01-16 20:50:03 UTC

## Contents

block_index . . . . .	2
compact . . . . .	3
echelon . . . . .	4
eliminate . . . . .	5

is_feasible . . . . .	7
is_totally_unimodular . . . . .	8
lintools . . . . .	9
normalize . . . . .	10
pinv . . . . .	11
project . . . . .	12
ranges . . . . .	14
sparse_constraints . . . . .	14
sparse_project . . . . .	16
subst_value . . . . .	18

**Index** **19**

---

block_index	<i>Find independent blocks of equations.</i>
-------------	--

---

**Description**

Find independent blocks of equations.

**Usage**

```
block_index(A, eps = 1e-08)
```

**Arguments**

A	[numeric] Matrix
eps	[numeric] Coefficients with absolute value < eps are treated as zero.

**Value**

A list containing numeric vectors, each vector indexing an independent block of rows in the system  $Ax \leq b$ .

**Examples**

```
A <- matrix(c(
  1,0,2,0,0,
  3,0,4,0,0,
  0,5,0,6,7,
  0,8,0,0,9
),byrow=TRUE,nrow=4)
b <- rep(0,4)
bi <- block_index(A)
lapply(bi,function(ii) compact(A[ii,,drop=FALSE],b=b[ii])$A)
```

---

compact	<i>Remove spurious variables and restrictions</i>
---------	---

---

### Description

A system of linear (in)equations can be compactified by removing zero-rows and zero-columns (=variables). Such rows and columns may arise after substitution (see [subst\\_value](#)) or elimination of a variable (see [eliminate](#)).

### Usage

```
compact(
  A,
  b,
  x = NULL,
  neq = nrow(A),
  nleq = 0,
  eps = 1e-08,
  remove_columns = TRUE,
  remove_rows = TRUE,
  deduplicate = TRUE,
  implied_equations = TRUE
)
```

### Arguments

A	[numeric] matrix
b	[numeric] vector
x	[numeric] vector
neq	[numeric] The first neq rows in A and b are treated as linear equalities.
nleq	[numeric] The nleq rows after neq are treated as inequations of the form $a \cdot x \leq b$ . All remaining rows are treated as strict inequations of the form $a \cdot x < b$ .
eps	[numeric] Anything with absolute value $< \text{eps}$ is considered zero.
remove_columns	[logical] Toggle remove spurious columns from A and variables from x
remove_rows	[logical] Toggle remove spurious rows
deduplicate	[logical] Toggle remove duplicate rows
implied_equations	[logical] replace cases of $a \cdot x \leq b$ and $a \cdot x \geq b$ with $a \cdot x = b$ .

### Value

A list with the following elements.

- A: The compactified version of input A

- `b`: The compactified version of input `b`
- `x`: The compactified version of input `x`
- `neq`: number of equations in new system
- `nleq`: number of inequations of the form  $a \cdot x \leq b$  in the new system
- `cols_removed`: [logical] indicates what elements of `x` (columns of `A`) have been removed

### Details

It is assumed that the system of equations is in normalized form (see [link{normalize}](#)).

---

echelon	<i>Reduced row echelon form</i>
---------	---------------------------------

---

### Description

Transform the equalities in a system of linear (in)equations or Reduced Row Echelon form (RRE)

### Usage

```
echelon(A, b, neq = nrow(A), nleq = 0, eps = 1e-08)
```

### Arguments

<code>A</code>	[numeric] matrix
<code>b</code>	[numeric] vector
<code>neq</code>	[numeric] The first <code>neq</code> rows of <code>A</code> , <code>b</code> are treated as equations.
<code>nleq</code>	[numeric] The <code>nleq</code> rows after <code>neq</code> are treated as inequations of the form $a \cdot x \leq b$ . All remaining rows are treated as strict inequations of the form $a \cdot x < b$ .
<code>eps</code>	[numeric] Values of magnitude less than <code>eps</code> are considered zero (for the purpose of handling machine rounding errors).

### Value

A list with the following components:

- `A`: the `A` matrix with equalities transformed to RRE form.
- `b`: the constant vector corresponding to `A`
- `neq`: the number of equalities in the resulting system.
- `nleq`: the number of inequalities of the form  $a \cdot x \leq b$ . This will only be passed to the output.

## Details

The parameters A, b and neq describe a system of the form  $Ax \leq b$ , where the first neq rows are equalities. The equalities are transformed to RRE form.

A system of equations is in **reduced row echelon** form when

- All zero rows are below the nonzero rows
- For every row, the leading coefficient (first nonzero from the left) is always right of the leading coefficient of the row above it.
- The leading coefficient equals 1, and is the only nonzero coefficient in its column.

## Examples

```
echelon(  
  A = matrix(c(  
    1,3,1,  
    2,7,3,  
    1,5,3,  
    1,2,0), byrow=TRUE, nrow=4)  
  , b = c(4,-9,1,8)  
  , neq=4  
)
```

---

eliminate

*Eliminate a variable from a set of edit rules*

---

## Description

Eliminating a variable amounts to deriving all (non-redundant) linear (in)equations not containing that variable. Geometrically, it can be interpreted as a projection of the solution space (vectors satisfying all equations) along the eliminated variable's axis.

## Usage

```
eliminate(  
  A,  
  b,  
  neq = nrow(A),  
  nleq = 0,  
  variable,  
  H = NULL,  
  h = 0,  
  eps = 1e-08  
)
```

**Arguments**

A	[numeric] Matrix
b	[numeric] vector
neq	[numeric] The first neq rows in A and b are treated as linear equalities.
nleq	[numeric] The nleq rows after neq are treated as inequations of the form $a \cdot x \leq b$ . All remaining rows are treated as strict inequations of the form $a \cdot x < b$ .
variable	[numeric logical character] Index in columns of A, representing the variable to eliminate.
H	[numeric] (optional) Matrix indicating how linear inequalities have been derived.
h	[numeric] (optional) number indicating how many variables have been eliminated from the original system using Fourier-Motzkin elimination.
eps	[numeric] Coefficients with absolute value $\leq$ eps are treated as zero.

**Value**

A list with the following components

- A: the A corresponding to the system with variables eliminated.
- b: the constant vector corresponding to the resulting system
- neq: the number of equations
- H: The memory matrix storing how each row was derived
- h: The number of variables eliminated from the original system.

**Details**

For equalities Gaussian elimination is applied. If inequalities are involved, Fourier-Motzkin elimination is used. In principle, FM-elimination can generate a large number of redundant inequations, especially when applied recursively. Redundancies can be recognized by recording how new inequations have been derived from the original set. This is stored in the H matrix when multiple variables are to be eliminated (Kohler, 1967).

**References**

D.A. Kohler (1967) Projections of convex polyhedral sets, Operational Research Center Report , ORC 67-29, University of California, Berkely.

H.P. Williams (1986) Fourier's method of linear programming and its dual. American Mathematical Monthly 93, pp 681-695.

**Examples**

```
# Example from Williams (1986)
A <- matrix(c(
  4, -5, -3, 1,
 -1, 1, -1, 0,
  1, 1, 2, 0,
```

```

-1, 0, 0, 0,
 0, -1, 0, 0,
 0, 0, -1, 0),byrow=TRUE,nrow=6)
b <- c(0,2,3,0,0,0)
L <- eliminate(A=A, b=b, neq=0, nleq=6, variable=1)

```

is\_feasible

*Check feasibility of a system of linear (in)equations***Description**

Check feasibility of a system of linear (in)equations

**Usage**

```
is_feasible(A, b, neq = nrow(A), nleq = 0, eps = 1e-08, method = "elimination")
```

**Arguments**

A	[numeric] matrix
b	[numeric] vector
neq	[numeric] The first neq rows in A and b are treated as linear equalities.
nleq	[numeric] The nleq rows after neq are treated as inequations of the form $a \cdot x \leq b$ . All remaining rows are treated as strict inequations of the form $a \cdot x < b$ .
eps	[numeric] Absolute values $< \text{eps}$ are treated as zero.
method	[character] At the moment, only the 'elimination' method is implemented.

**Examples**

```

# An infeasible system:
# x + y == 0
# x > 0
# y > 0
A <- matrix(c(1,1,1,0,0,1),byrow=TRUE,nrow=3)
b <- rep(0,3)
is_feasible(A=A,b=b,neq=1,nleq=0)

# A feasible system:
# x + y == 0
# x >= 0
# y >= 0
A <- matrix(c(1,1,1,0,0,1),byrow=TRUE,nrow=3)
b <- rep(0,3)
is_feasible(A=A,b=b,neq=1,nleq=2)

```

---

is\_totally\_unimodular *Test for total unimodularity of a matrix.*

---

### Description

Check whether a matrix is totally unimodular.

### Usage

```
is_totally_unimodular(A)
```

### Arguments

A                    An object of class `matrix`.

### Details

A matrix for which the determinant of every square submatrix equals  $-1$ ,  $0$  or  $1$  is called **totally unimodular**. This function tests whether a matrix with coefficients in  $\{-1, 0, 1\}$  is totally unimodular. It tries to reduce the matrix using the reduction method described in Scholtus (2008). Next, a test based on Heller and Tompkins (1956) or Raghavachari is performed.

### Value

logical

### References

Heller I and Tompkins CB (1956). An extension of a theorem of Dantzig's In Kuhn HW and Tucker AW (eds.), pp. 247-254. Princeton University Press.

Raghavachari M (1976). A constructive method to recognize the total unimodularity of a matrix. *\_Zeitschrift fur operations research\_*, \*20\*, pp. 59-61.

Scholtus S (2008). Algorithms for correcting some obvious inconsistencies and rounding errors in business survey data. Technical Report 08015, Netherlands.

### Examples

```
# Totally unimodular matrix, reduces to nothing
A <- matrix(c(
  1,1,0,0,0,
  -1,0,0,1,0,
  0,0,0,1,1,0,
  0,0,0,-1,1),nrow=5)
is_totally_unimodular(A)

# Totally unimodular matrix, by Heller-Tompson criterium
A <- matrix(c(
  1,1,0,0,
```



```
0,0,1,1,
1,0,1,0,
0,1,0,1),nrow=4)
is_totally_unimodular(A)

# Totally unimodular matrix, by Raghavachani recursive criterium
A <- matrix(c(
  1,1,1,
  1,1,0,
  1,0,1))
is_totally_unimodular(A)
```

---

lintoos

*Tools for manipulating linear systems of (in)equations*

---

## Description

This package offers a basic and consistent interface to a number of operations on linear systems of (in)equations not available in base R. Except for the projection on the convex polytope, operations are currently supported for dense matrices only.

## Details

The following operations are implemented.

- Split matrices in independent blocks
- Remove spurious rows and columns from a system of (in)equations
- Rewrite equalities in reduced row echelon form
- Eliminate variables through Gaussian or Fourier-Motzkin elimination
- Determine the feasibility of a system of linear (in)equations
- Compute Moore-Penrose Pseudoinverse
- Project a vector onto the convec polytope described by a set of linear (in)equations
- Simplify a system by substituting values

Most functions assume a system of (in)equations to be stored in a standard form. The [normalize](#) function can bring any system of equations to this form.

---

 normalize

*Bring a system of (in)equalities in a standard form*


---

### Description

Bring a system of (in)equalities in a standard form

### Usage

```
normalize(A, b, operators, unit = 0)
```

### Arguments

A	[numeric] Matrix
b	[numeric] vector
operators	[character] operators in {<, <=, ==, >=, >}.
unit	[numeric] (nonnegative) Your unit of measurement. This is used to replace strict inequations of the form $a \cdot x < b$ with $a \cdot x \leq b - \text{unit}$ . Typically, unit is related to the units in which your data is measured. If unit is 0, inequations are not replaced.

### Value

A list with the following components

- A: the A corresponding to the normalized system.
- b: the constant vector corresponding to the normalized system
- neq: the number of equations
- nleq: the number of non-strict inequations ( $\leq$ )
- order: the index vector used to permute the original rows of A.

### Details

For this package, a set of equations is in normal form when

- The first neq rows represent linear equalities.
- The next nleq rows represent inequalities of the form  $a \cdot x \leq b$
- All other rows are strict inequalities of the form  $a \cdot x < b$

If  $\text{unit} > 0$ , the strict inequalities  $a \cdot x < b$  are replaced with inequations of the form  $a \cdot x \leq b - \text{unit}$ , where unit represents the precision of measurement.

**Examples**

```
A <- matrix(1:12,nrow=4)
b <- 1:4
ops <- c("<=", "==", "==" , "<")
normalize(A,b,ops)
normalize(A,b,ops,unit=0.1)
```

pinv

*Moore-Penrose pseudoinverse***Description**

Compute the pseudoinverse of a matrix using the SVD-construction

**Usage**

```
pinv(A, eps = 1e-08)
```

**Arguments**

A	[numeric] matrix
eps	[numeric] tolerance for determining zero singular values

**Details**

The Moore-Penrose pseudoinverse (sometimes called the generalized inverse)  $A^+$  of a matrix  $A$  has the property that  $A^+AA^+ = A$ . It can be constructed as follows.

- Compute the singular value decomposition  $A = UDV^T$
- Replace diagonal elements in  $D$  of which the absolute values are larger than some limit eps with their reciprocal values
- Compute  $A^+ = UDV^T$

**References**

S Lipshutz and M Lipson (2009) Linear Algebra. In: Schuam's outlines. McGraw-Hill

**Examples**

```
A <- matrix(c(
  1, 1, -1, 2,
  2, 2, -1, 3,
  -1, -1, 2, -3
),byrow=TRUE,nrow=3)
# multiply by 55 to retrieve whole numbers
pinv(A) * 55
```

---

project	<i>Project a vector on the border of the region defined by a set of linear (in)equality restrictions.</i>
---------	---

---

### Description

Compute a vector, closest to  $x$  in the Euclidean sense, satisfying a set of linear (in)equality restrictions.

### Usage

```
project(
  x,
  A,
  b,
  neq = length(b),
  w = rep(1, length(x)),
  eps = 0.01,
  maxiter = 1000L
)
```

### Arguments

<code>x</code>	[numeric] Vector that needs to satisfy the linear restrictions.
<code>A</code>	[matrix] Coefficient matrix for linear restrictions.
<code>b</code>	[numeric] Right hand side of linear restrictions.
<code>neq</code>	[numeric] The first <code>neq</code> rows in <code>A</code> and <code>b</code> are treated as linear equalities. The others as Linear inequalities of the form $Ax \leq b$ .
<code>w</code>	[numeric] Optional weight vector of the same length as <code>x</code> . Must be positive.
<code>eps</code>	The maximum allowed deviation from the constraints (see details).
<code>maxiter</code>	maximum number of iterations

### Value

A list with the following entries:

- `x`: the adjusted vector
- `status`: Exit status:
  - 0: success
  - 1: could not allocate enough memory (space for approximately  $2(m + n)$  doubles is necessary).
  - 2: divergence detected (set of restrictions may be contradictory)
  - 3: maximum number of iterations reached
- `eps`: The tolerance achieved after optimizing (see Details).

- iterations: The number of iterations performed.
- duration: the time it took to compute the adjusted vector
- objective: The (weighted) Euclidean distance between the initial and the adjusted vector

## Details

The tolerance `eps` is defined as the maximum absolute value of the difference vector  $Ax - b$  for equalities. For inequalities, the difference vector is set to zero when it's value is lesser than zero (i.e. when the restriction is satisfied). The algorithm iterates until either the tolerance is met, the number of allowed iterations is exceeded or divergence is detected.

## See Also

[sparse\\_project](#)

## Examples

```
# the system
# x + y = 10
# -x <= 0 # ==> x > 0
# -y <= 0 # ==> y > 0
#
A <- matrix(c(
  1,1,
  -1,0,
  0,-1), byrow=TRUE, nrow=3
)
b <- c(10,0,0)

# x and y will be adjusted by the same amount
project(x=c(4,5), A=A, b=b, neq=1)

# One of the inequalities violated
project(x=c(-1,5), A=A, b=b, neq=1)

# Weighted distances: 'heavy' variables change less
project(x=c(4,5), A=A, b=b, neq=1, w=c(100,1))

# if w=1/x0, the ratio between coefficients of x0 stay the same (to first order)
x0 <- c(x=4,y=5)
x1 <- project(x=x0,A=A,b=b,neq=1,w=1/x0)

x0[1]/x0[2]
x1$x[1] / x1$x[2]
```

---

ranges	<i>Derive variable ranges from linear restrictions</i>
--------	--

---

### Description

Gaussian and/or Fourier-Motzkin elimination is used to derive upper and lower limits implied by a system of (in)equations.

### Usage

```
ranges(A, b, neq = nrow(A), nleq = 0, eps = 1e-08)
```

### Arguments

A	[numeric] Matrix
b	[numeric] vector
neq	[numeric] The first neq rows in A and b are treated as linear equalities.
nleq	[numeric] The nleq rows after neq are treated as inequations of the form $a \cdot x \leq b$ . All remaining rows are treated as strict inequations of the form $a \cdot x < b$ .
eps	[numeric] Coefficients with absolute value $\leq$ eps are treated as zero. using Fourier-Motzkin elimination.

---

sparse_constraints	<i>Generate sparse set of constraints.</i>
--------------------	--

---

### Description

Generate a constraint set to be used by [sparse\\_project](#)

### Usage

```
sparse_constraints(object, ...)

sparseConstraints(object, ...)

## S3 method for class 'data.frame'
sparse_constraints(object, b, neq = length(b), base = 1L, sorted = FALSE, ...)

## S3 method for class 'sparse_constraints'
print(x, range = 1L:10L, ...)
```

**Arguments**

object	R object to be translated to sparse_constraints format.
...	options to be passed to other methods
b	Constant vector
neq	The first new equations are interpreted as equality constraints, the rest as '<='
base	are the indices in object[, 1:2] base 0 or base 1?
sorted	is object sorted by the first column?
x	an object of class sparse_constraints
range	integer vector stating which constraints to print

**Value**

Object of class sparse\_constraints (see details).

**Note**

As of version 0.1.1.0, sparseConstraints is deprecated. Use sparse\_constraints instead.

**Details**

The sparse\_constraints objects holds coefficients of  $\mathbf{A}$  and  $\mathbf{b}$  of the system  $\mathbf{Ax} \leq \mathbf{b}$  in sparse format, outside of R's memory. It can be reused to find solutions for vectors to adjust.

In R, it is a *reference object*. In particular, it is meaningless to

- Copy the object. You only will only generate a pointer to physically the same object.
- Save the object. The physical object is destroyed when R closes, or when R's garbage collector cleans up a removed sparse\_constraints object.

**The \$project method**

Once a sparse\_constraints object sc is created, you can reuse it to optimize several vectors by calling sc\$project() with the following parameters:

- x: [numeric] the vector to be optimized
- w: [numeric] the weight vector (of length(x)). By default all weights equal 1.
- eps: [numeric] desired tolerance. By default  $10^{-2}$
- maxiter: [integer] maximum number of iterations. By default 1000.

The return value of \$spa is the same as that of [sparse\\_project](#).

**See Also**

[sparse\\_project](#), [project](#)

**Examples**

```

# The following system of constraints, stored in
# row-column-coefficient format
#
# x1 + x8 == 950,
# x3 + x4 == 950 ,
# x6 + x7 == x8,
# x4 > 0
#
A <- data.frame(
  row = c( 1, 1, 2, 2, 3, 3, 3, 4)
  , col = c( 1, 2, 3, 4, 2, 5, 6, 4)
  , coef = c(-1,-1,-1,-1, 1,-1,-1,-1)
)
b <- c(-950, -950, 0,0)

sc <- sparse_constraints(A, b, neq=3)

# Adjust the 0-vector minimally so all constraints are met:
sc$project(x=rep(0,8))

# Use the same object to adjust the 100*1-vector
sc$project(x=rep(100,8))

# use the same object to adjust the 0-vector, but with different weights
sc$project(x=rep(0,8),w=1:8)

```

---

 sparse\_project

*Successive projections with sparsely defined restrictions*


---

**Description**

Compute a vector, closest to  $x$  satisfying a set of linear (in)equality restrictions.

**Usage**

```

sparse_project(
  x,
  A,
  b,
  neq = length(b),
  w = rep(1, length(x)),
  eps = 0.01,
  maxiter = 1000L,
  ...
)

```



**Arguments**

x	[numeric] Vector to optimize, starting point.
A	[data.frame] Coefficient matrix in [row,column,coefficient] format.
b	[numeric] Constant vector of the system $Ax \leq b$
neq	[integer] Number of equalities
w	[numeric] weight vector of same length of x
eps	maximally allowed tolerance
maxiter	maximally allowed number of iterations.
...	extra parameters passed to <a href="#">sparse_constraints</a>

**Value**

A list with the following entries:

- x: the adjusted vector
- status: Exit status:
  - 0: success
  - 1: could not allocate enough memory (space for approximately  $2(m + n)$  doubles is necessary).
  - 2: divergence detected (set of restrictions may be contradictory)
  - 3: maximum number of iterations reached
- eps: The tolerance achieved after optimizing (see Details).
- iterations: The number of iterations performed.
- duration: the time it took to compute the adjusted vector
- objective: The (weighted) Euclidean distance between the initial and the adjusted vector

**Details**

The tolerance eps is defined as the maximum absolute value of the difference vector  $Ax - b$  for equalities. For inequalities, the difference vector is set to zero when it's value is lesser than zero (i.e. when the restriction is satisfied). The algorithm iterates until either the tolerance is met, the number of allowed iterations is exceeded or divergence is detected.

**See Also**

[project](#), [sparse\\_constraints](#)

**Examples**

```
# the system
# x + y = 10
# -x <= 0 # ==> x > 0
# -y <= 0 # ==> y > 0
# Defined in the row-column-coefficient form:
```

```
A <- data.frame(
  row = c(1,1,2,3)
  , col = c(1,2,1,2)
  , coef= c(1,1,-1,-1)
)
b <- c(10,0,0)

sparse_project(x=c(4,5),A=A,b=b)
```

---

 subst\_value

---

*Substitute a value in a system of linear (in)equations*


---

### Description

Substitute a value in a system of linear (in)equations

### Usage

```
subst_value(A, b, variables, values, remove_columns = FALSE, eps = 1e-08)
```

### Arguments

A	[numeric] matrix
b	[numeric] vector
variables	[numeric logical character] vector of column indices in A
values	[numeric] vector of values to substitute.
remove_columns	[logical] Remove spurious columns when substituting?
eps	[numeric] scalar. Any value with absolute value below eps will be interpreted as zero.

### Value

A list with the following components:

- A: the A corresponding to the simplified system.
- b: the constant vector corresponding to the new system

### Details

A system of the form  $Ax \leq b$  can be simplified if one or more of the  $x[i]$  values is fixed.

# Index

block\_index, 2

compact, 3

echelon, 4

eliminate, 3, 5

is\_feasible, 7

is\_totally\_unimodular, 8

lintools, 9

matrix, 8

normalize, 9, 10

pinv, 11

print.sparse\_constraints  
    (sparse\_constraints), 14

project, 12, 15, 17

ranges, 14

sparse\_constraints, 14, 17

sparse\_project, 13–15, 16

sparseConstraints (sparse\_constraints),  
    14

subst\_value, 3, 18