Package ‘ROI’

April 20, 2023

Version 1.0-1

Title R Optimization Infrastructure

Description The R Optimization Infrastructure ('ROI') <doi:10.18637/jss.v094.i15> is a sophisticated framework for handling optimization problems in R. Additional information can be found on the 'ROI' homepage <https://roi.r-forge.r-project.org/>.

Depends R (>= 2.10)

Imports methods, registry (>= 0.5), slam, utils, checkmate

Suggests numDeriv

License GPL-3

Encoding UTF-8

URL https://roi.r-forge.r-project.org/,
https://r-forge.r-project.org/projects/roi/

RoxygenNote 7.2.3

NeedsCompilation no

Author Kurt Hornik [aut] (<https://orcid.org/0000-0003-4198-9911>),
David Meyer [aut],
Florian Schwendinger [aut] (<https://orcid.org/0000-0002-3983-9773>),
Stefan Theussl [aut, cre] (<https://orcid.org/0000-0002-6523-4620>),
Diethelm Wuertz [ctb]

Maintainer Stefan Theussl <Stefan.Theussl@R-Project.org>

Repository CRAN

Date/Publication 2023-04-20 19:42:30 UTC

R topics documented:

`as.L_term` *Canonicalize the Linear Term*

Description

Canonicalize the linear term of a linear constraint. Objects from the following classes can be canonicalized: `"NULL"`, `"numeric"`, `"matrix"`, `"simple_triplet_matrix"` and `"list"`.

Usage

```
as.L_term(x, ...)
```

Arguments

`x`	an R object.
`...`	further arguments passed to or from other methods.

Details

In the case of `lists` `"as.Q_term"` is applied to every element of the list, for `NULL` one can supply the optional arguments `"nrow"` and `"ncol"` which will create a `"simple_triplet_zero_matrix"` with the specified dimension.

Value

an object of class `"simple_triplet_matrix"`

`as.Q_term` *Canonicalize the Quadraric Term*

Description

Canonicalize the quadraric term of a quadratic constraint. Objects from the following classes can be canonicalized: `"NULL"`, `"numeric"`, `"matrix"`, `"simple_triplet_matrix"` and `"list"`.

Usage

```
as.Q_term(x, ...)

## S3 method for class 'list'
as.Q_term(x, ...)

## S3 method for class 'numeric'
as.Q_term(x, ...)

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

## S3 method for class 'simple_triplet_matrix'
as.Q_term(x, ...)

## S3 method for class '`NULL`'
as.Q_term(x, ...)
```

Arguments

`x`	an R object.
`...`	further arguments

Details

In the case of `lists` "`as.Q_term`" is applied to every element of the list, for `NULL` one can supply the optional arguments "`nrow`" and "`ncol`" which will create a "`simple_triplet_zero_matrix`" with the specified dimension.

Value

an object of class "`simple_triplet_matrix`"

`bound (Constructors)` *bound*

Description

ROI distinguishes between 2 different types of bounds:

- No Bounds `NO_bound`
- Variable Bounds `V_bound` (inherits from "`bound`")

Usage

```
## S3 method for class 'bound'
c(...)

is.bound(x)
```

Arguments

`x`	object to be tested
`...`	arguments (inheriting from bound) to be combined

Details

ROI provides the method `V_bound` as constructor for variable bounds. `NO_bound` is not explicitly implemented but represented by `NULL`.

`bounds (Set/Get)` *Bounds - Accessor and Mutator Functions*

Description

The bounds of a given optimization problem (OP) can be accessed or mutated via the method `'bounds'`.

Usage

```
bounds(x)

## S3 method for class 'OP'
bounds(x)

bounds(x) <- value
```

Arguments

`x`	an object of type `'OP'` used to select the method.
`value`	an object derived from `'bound'` (`'V_bound'`) or `NULL`.

Value

the extracted bounds object on get and the altered `'OP'` object on set.

Examples

```
## Not run:
lp_obj <- L_objective(c(1, 2))
lp_con <- L_constraint(c(1, 1), dir="==", rhs=2)
lp_bound <- V_bound(ui=1:2, ub=c(3, 3))
lp <- OP(objective=lp_obj, constraints=lp_con, bounds=lp_bound, maximum=FALSE)
bounds(lp)
x <- ROI_solve(lp)
x$objval
x$solution
bounds(lp) <- V_bound(ui=1:2, ub=c(1, 1))
y <- ROI_solve(lp)
```

```
y$objval
y$solution

## End(Not run)
```

---

constraint (Constructors)
constraint

Description

ROI distinguishes between 5 different types of constraint:

- No Constraint `NO_constraint` (inherits from "constraint")
- Linear Constraint `L_constraint` (inherits from "constraint")
- Quadratic Constraint `Q_constraint` (inherits from "constraint")
- Conic Constraint `C_constraint` (inherits from "constraint")
- Function Constraint `F_constraint` (inherits from "constraint")

Usage

```
## S3 method for class 'constraint'
c(..., recursive = FALSE)

as.constraint(x)

is.constraint(x)

## S3 method for class 'constraint'
dim(x)
```

Arguments

`recursive`	a logical, giving if the arguments should be combined recursively.
`x`	an object to be coerced or tested.
`...`	objects to be combined.

constraint directions *Replicate "==", ">=" and "<=" Signs*

Description

The utility functions `eq`, `leq` and `geq` replicate the signs `"=="`, `">="` and `"<="` `n` times.

Usage

```
eq(n)

leq(n)

geq(n)
```

Arguments

`n`	an integer giving the number of times the sign should be repeated.

Examples

```
eq(3)
leq(2)
geq(4)
```

constraints (Set/Get) *Constraints - Accessor and Mutator Functions*

Description

The constraints of a given optimization problem (OP) can be accessed or mutated via the method `'constraints'`.

Usage

```
constraints(x)

## S3 method for class 'OP'
constraints(x)

constraints(x) <- value
```

Arguments

`x`	an object used to select the method.
`value`	an R object.

Value

the extracted constraints object.

Author(s)

Stefan Theussl

Examples

```
## minimize: x + 2 y
## subject to: x + y >= 1
## x, y >= 0
x <- OP(1:2)
constraints(x) <- L_constraint(c(1, 1), ">=", 1)
constraints(x)
```

`C_constraint` *Conic Constraints*

Description

Conic constraints are often written in the form

$$Lx + s = rhs$$

where L is a $m \times n$ (sparse) matrix and $s \in \mathcal{K}$ are the slack variables restricted to some cone $\mathcal{K}$ which is typically the product of simpler cones $\mathcal{K} = \prod \mathcal{K}_i$. The right hand side rhs is a vector of length m.

Usage

```
C_constraint(L, cones, rhs, names = NULL)

as.C_constraint(x, ...)

is.C_constraint(x)

## S3 method for class 'C_constraint'
length(x)

## S3 method for class 'C_constraint'
variable.names(object, ...)

## S3 method for class 'C_constraint'
terms(x, ...)
```

Arguments

`L`	a numeric vector of length n (a single constraint) or a matrix of dimension $m \times n$, where n is the number of objective variables and m is the number of constraints. Matrices can be of class `"simple_triplet_matrix"` to allow a sparse representation of constraints.
`cones`	an object of class `"cone"` created by the combination, of `K_zero`, `K_lin`, `K_soc`, `K_psd`, `K_expp`, `K_expd`, `K_powp` or `K_powd`.
`rhs`	a numeric vector giving the right hand side of the constraints.
`names`	an optional character vector giving the names of x (column names of L).
`x`	an R object.
`...`	further arguments passed to or from other methods (currently ignored).
`object`	an R object.

Value

an object of class `"C_constraint"` which inherits from `"constraint"`.

Examples

```
## minimize:  x1 + x2 + x3
## subject to:
##    x1 == sqrt(2)
##    ||(x2, x3)|| <= x1
x <- OP(objective = c(1, 1, 1),
        constraints = C_constraint(L = rbind(rbind(c(1, 0, 0)),
                                             diag(x=-1, 3)),
                                   cones = c(K_zero(1), K_soc(3)),
                                   rhs = c(sqrt(2), rep(0, 3))),
        types = rep("C", 3),
        bounds =  V_bound(li = 1:3, lb = rep(-Inf, 3)), maximum = FALSE)
```

`equal` *Compare two Objects*

Description

The utility function `equal` can be used to compare two **ROI** objects and is mainly used for testing purposes.

Usage

```
equal(x, y, ...)

## S3 method for class '`NULL`'
equal(x, y, ...)
```

```
## S3 method for class 'logical'
equal(x, y, ...)

## S3 method for class 'integer'
equal(x, y, ...)

## S3 method for class 'numeric'
equal(x, y, ...)

## S3 method for class 'character'
equal(x, y, ...)

## S3 method for class 'list'
equal(x, y, ...)

## S3 method for class 'simple_triplet_matrix'
equal(x, y, ...)

## S3 method for class 'L_constraint'
equal(x, y, ...)

## S3 method for class 'Q_constraint'
equal(x, y, ...)

## S3 method for class 'V_bound'
equal(x, y, ...)
```

Arguments

x	an R object to be compared with object y.
y	an R object to be compared with object x.
...	optional arguments to equal.

Value

TRUE if x and y are equal FALSE otherwise.

Examples

```
## compare numeric values
equal(1e-4, 1e-5, tol=1e-3)
## L_constraint
lc1 <- L_constraint(diag(1), dir=c("=="), rhs=1)
lc2 <- L_constraint(diag(2), dir=c("==", "<="), rhs=1:2)
equal(lc1, lc1)
equal(lc1, lc2)
```

F_constraint *Function Constraints*

Description

Function (or generally speaking nonlinear) constraints are typically of the form

$$f(x) \leq b$$

where $f()$ is a well-defined R function taking the objective variables x (typically a numeric vector) as arguments. b is called the right hand side of the constraints.

Usage

```
F_constraint(F, dir, rhs, J = NULL, names = NULL)

## S3 method for class 'F_constraint'
variable.names(object, ...)

is.F_constraint(x)

as.F_constraint(x, ...)

## S3 method for class '`NULL`'
as.F_constraint(x, ...)

## S3 method for class 'NO_constraint'
as.F_constraint(x, ...)

## S3 method for class 'constraint'
as.F_constraint(x, ...)

## S3 method for class 'F_constraint'
terms(x, ...)
```

Arguments

F	a `function` or a list of `functions` of length m. Each `function` takes n parameters as input and must return a scalar. Thus, n is the number of objective variables and m is the number of constraints.
dir	a character vector with the directions of the constraints. Each element must be one of "<=", ">=" or "==".
rhs	a numeric vector with the right hand side of the constraints.
J	an optional `function` holding the Jacobian of F.
names	an optional character vector giving the names of x.
object	an R object.
x	object to be tested.
...	further arguments passed to or from other methods (currently ignored).

Value

an object of class "F_constraint" which inherits from "constraint".

Author(s)

Stefan Theussl

F_objective — *General (Nonlinear) Objective Function*

Description

General objective function $f(x)$ to be optimized.

Usage

```
F_objective(F, n, G = NULL, H = NULL, names = NULL)

## S3 method for class 'F_objective'
terms(x, ...)

as.F_objective(x)

## S3 method for class 'F_objective'
variable.names(object, ...)
```

Arguments

F	an R "function" taking a numeric vector x of length n as argument.
n	the number of objective variables.
G	an R "function" returning the gradient at x.
H	an optional function holding the Hessian of F.
names	an optional character vector giving the names of x.
x	an R object.
...	further arguments passed to or from other methods
object	an R object.

Value

an object of class "F_objective" which inherits from "objective".

Author(s)

Stefan Theussl

G *Extract Gradient information*

Description

Extract the gradient from its argument (typically a ROI object of class "objective").

Usage

```
G(x, ...)
```

Arguments

x	an object used to select the method.
...	further arguments passed down to the grad() function for calculating gradients (only for "F_objective").

Details

By default **ROI** uses the "grad" function from the **numDeriv** package to derive the gradient information. An alternative function can be provided via "ROI_options". For example ROI_options("gradient", myGrad) would tell **ROI** to use the function "myGrad" for the gradient calculation. The only requirement to the function "myGrad" is that it has the argument "func" which takes a function with a scalar real result.

Value

a "function".

Examples

```
## Not run:
   f <- function(x) {
       return( 100 * (x[2] - x[1]^2)^2 + (1 - x[1])^2 )
   }
   x <- OP( objective = F_objective(f, n=2L),
            bounds = V_bound(li=1:2, ui=1:2, lb=c(-3, -3), ub=c(3, 3)) )
   G(objective(x))(c(0, 0)) ## gradient numerically approximated by numDeriv


   f.gradient <- function(x) {
       return( c( -400 * x[1] * (x[2] - x[1] * x[1]) - 2 * (1 - x[1]),
                  200 * (x[2] - x[1] * x[1])) )
   }
   x <- OP( objective = F_objective(f, n=2L, G=f.gradient),
            bounds = V_bound(li=1:2, ui=1:2, lb=c(-3, -3), ub=c(3, 3)) )
   G(objective(x))(c(0, 0)) ## gradient calculated by f.gradient

## End(Not run)
```

`is.default_bound` *Check for default bounds*

Description

tests if the given object is an variable bound which represents default values only (i.e., all lower bounds are `0` and all upper bounds as `Inf`).

Usage

```
is.default_bound(x)
```

Arguments

`x`	object to be tested

Value

a logical of length one indicating wether default bounds are given

`J` *Extract Jacobian Information*

Description

Derive the Jacobian for a given constraint.

Usage

```
J(x, ...)

## S3 method for class 'L_constraint'
J(x, ...)

## S3 method for class 'Q_constraint'
J(x, ...)
```

Arguments

`x`	a `L_constraint`, `Q_constraint` or `F_constraint`.
`...`	further arguments

Value

a list of functions

Examples

```
L <- matrix(c(3, 4, 2, 2, 1, 2, 1, 3, 2), nrow=3, byrow=TRUE)
lc <- L_constraint(L = L, dir = c("<=", "<=", "<="), rhs = c(60, 40, 80))
J(lc)
```

K_zero *Cone Constructors*

Description

Constructor functions for the different cone types. Currently **ROI** supports eight different types of cones.

- `Zero cone`
$$\mathcal{K}_{\text{zero}} = \{0\}$$
- `Nonnegative (linear) cone`
$$\mathcal{K}_{\text{lin}} = \{x | x \geq 0\}$$
- `Second-order cone`
$$\mathcal{K}_{\text{soc}} = \{(t, x) \mid ||x||_2 \leq t, x \in R^n, t \in R\}$$
- `Positive semidefinite cone`
$$\mathcal{K}_{\text{psd}} = \left\{X \mid min(eig(X)) \geq 0,\ X = X^T,\ X \in R^{n \times n}\right\}$$
- `Exponential cone`
$$\mathcal{K}_{\text{expp}} = \left\{(x, y, z) \mid ye^{\frac{x}{y}} \leq z,\ y > 0\right\}$$
- `Dual exponential cone`
$$\mathcal{K}_{\text{expd}} = \left\{(u, v, w) \mid\ -ue^{\frac{v}{u}} \leq ew, u < 0\right\}$$
- `Power cone`
$$\mathcal{K}_{\text{powp}} = \left\{(x, y, z) \mid x^{\alpha} * y^{(1-\alpha)} \geq |z|,\ x \geq 0,\ y \geq 0\right\}$$
- `Dual power cone`
$$\mathcal{K}_{\text{powd}} = \left\{(u, v, w) \mid\ \left(\frac{u}{\alpha}\right)^{\alpha} * \left(\frac{v}{(1-\alpha)}\right)^{(1-\alpha)} \geq |w|,\ u \geq 0,\ v \geq 0\right\}$$

Usage

```
K_zero(size)

K_lin(size)

K_soc(sizes)

K_psd(sizes)

K_expp(size)

K_expd(size)

K_powp(alpha)

K_powd(alpha)
```

Arguments

`size`	a integer giving the size of the cone, if the dimension of the cones is fixed (i.e. `zero`, `lin`, `expp`, `expd`) the number of cones is sufficient to define the dimension of the product cone.
`sizes`	a integer giving the sizes of the cones, if the dimension of the cones is not fixed (i.e. `soc`, `psd`) we have to define the sizes of each single cone.
`alpha`	a numeric vector giving the `alphas` for the (dual) power cone.

Examples

```
K_zero(3) ## 3 equality constraints
K_lin(3)  ## 3 constraints where the slack variable s lies in the linear cone
```

`L_constraint` *Linear Constraints*

Description

Linear constraints are typically of the form

$$Lx \leq rhs$$

where L is a $m \times n$ (sparse) matrix of coefficients to the objective variables x and the right hand side rhs is a vector of length m.

Usage

```
L_constraint(L, dir, rhs, names = NULL)

## S3 method for class 'L_constraint'
variable.names(object, ...)

as.L_constraint(x, ...)

is.L_constraint(x)

## S3 method for class 'L_constraint'
length(x)

## S3 method for class 'L_constraint'
terms(x, ...)
```

Arguments

`L`	a numeric vector of length n (a single constraint) or a matrix of dimension $m \times n$, where n is the number of objective variables and m is the number of constraints. Matrices can be of class "`simple_triplet_matrix`" to allow a sparse representation of constraints.
`dir`	a character vector with the directions of the constraints. Each element must be one of "`<=`", "`>=`" or "`==`".
`rhs`	a numeric vector with the right hand side of the constraints.
`names`	an optional character vector giving the names of x (column names of A).
`object`	an R object.
`...`	further arguments passed to or from other methods (currently ignored).
`x`	an R object.

Value

an object of class "`L_constraint`" which inherits from "`constraint`".

Author(s)

Stefan Theussl

`L_objective` *Linear Objective Function*

Description

A linear objective function is typically of the form

$$c^\top x$$

where c is a (sparse) vector of coefficients to the n objective variables x.

Usage

```
L_objective(L, names = NULL)

## S3 method for class 'L_objective'
terms(x, ...)

as.L_objective(x)

## S3 method for class 'L_objective'
variable.names(object, ...)
```

Arguments

`L`	a numeric vector of length n or an object of class `"simple_triplet_matrix"` (or coercible to such) with dimension $1 \times n$, where n is the number of objective variables. Names will be preserved and used e.g., in the print method.
`names`	an optional character vector giving the names of x (column names of L).
`x`	an R object.
`...`	further arguments passed to or from other methods
`object`	an R object.

Value

an object of class `"L_objective"` which inherits from `"Q_objective"` and `"objective"`.

Author(s)

Stefan Theussl

`maximum (Set/Get)` *Maximum - Accessor and Mutator Functions*

Description

The maximum of a given optimization problem (OP) can be accessed or mutated via the method `'maximum'`. If `'maximum'` is set to `TRUE` the OP is maximized, if `'maximum'` is set to `FALSE` the OP is minimized.

Usage

```
maximum(x)

maximum(x) <- value
```

Arguments

`x`	an object used to select the method.
`value`	an R object.

Value

a logical giving the direction.

Examples

```
## maximize: x + y
## subject to: x + y <= 2
## x, y >= 0
x <- OP(objective = c(1, 1),
        constraints = L_constraint(L = c(1, 1), dir = "<=", rhs = 2),
        maximum = FALSE)
maximum(x) <- TRUE
maximum(x)
```

---

nlminb2 — *Nonlinear programming with nonlinear constraints.*

Description

This function was contributed by Diethelm Wuertz.

Usage

```
nlminb2(
  start,
  objective,
  eqFun = NULL,
  leqFun = NULL,
  lower = -Inf,
  upper = Inf,
  gradient = NULL,
  hessian = NULL,
  control = list()
)
```

Arguments

`start`	numeric vector of start values.
`objective`	the function to be minimized $f(x)$.
`eqFun`	functions specifying equal constraints of the form $h_i(x) = 0$. Default: `NULL` (no equal constraints).

leqFun	functions specifying less equal constraints of the form $g_i(x) <= 0$. Default: NULL (no less equal constraints).
lower	a numeric representing lower variable bounds. Repeated as needed. Default: -Inf.
upper	a numeric representing upper variable bounds. Repeated as needed. Default: Inf.
gradient	gradient of $f(x)$. Default: NULL (no gradiant information).
hessian	hessian of $f(x)$. Default: NULL (no hessian provided).
control	a list of control parameters. See nlminb() for details. The parameter "scale" is set here in contrast to nlminb() .

Value

list()

Author(s)

Diethelm Wuertz

Examples

```
## Equal constraint function
eval_g0_eq <- function( x, params = c(1,1,-1)) {
       return( params[1]*x^2 + params[2]*x + params[3] )
   }
eval_f0 <- function( x, ... ) {
       return( 1 )
   }
```

NO_constraint   *Class:* "NO_constraint"

Description

In case the constraints slot in the problem object is NULL the return value of a call of constraints() will return an object of class "NO_constraint" which inherits from "L_constraint".

Usage

```
NO_constraint(n_obj)

as.NO_constraint(x, ...)

is.NO_constraint(x)
```

Arguments

`n_obj`	a numeric vector of length 1 representing the number of objective variables.
`x`	an R object.
`...`	further arguments passed to or from other methods (currently ignored).

Value

an object of class `"NO_constraint"` which inherits from `"L_constraint"` and `"constraint"`.

Author(s)

Stefan Theussl

`objective (Set/Get)` *Objective - Accessor and Mutator Functions*

Description

The objective of a given optimization problem (OP) can be accessed or mutated via the method `'objective'`.

Usage

```
objective(x)

objective(x) <- value

as.objective(x)
```

Arguments

`x`	an object used to select the method.
`value`	an R object.

Value

a function inheriting from `"objective"`.

Author(s)

Stefan Theussl

Examples

```
x <- OP()
objective(x) <- 1:3
```

OP — *Optimization Problem Constructor*

Description

Optimization problem constructor

Usage

```
OP(objective, constraints, types, bounds, maximum = FALSE)

as.OP(x)
```

Arguments

`objective`	an object inheriting from class `"objective"`.
`constraints`	an object inheriting from class `"constraints"`.
`types`	a character vector giving the types of the objective variables, with `"C"`, `"I"`, and `"B"` corresponding to continuous, integer, and binary, respectively, or `NULL` (default), taken as all-continuous. Recycled as needed.
`bounds`	`NULL` (default) or a list with elements `upper` and `lower` containing the indices and corresponding bounds of the objective variables. The default for each variable is a bound between 0 and `Inf`.
`maximum`	a logical giving the direction of the optimization. `TRUE` means that the objective is to maximize the objective function, `FALSE` (default) means to minimize it.
`x`	an R object.

Value

an object of class `"OP"`.

Author(s)

Stefan Theussl

References

Theussl S, Schwendinger F, Hornik K (2020). 'ROI: An Extensible R Optimization Infrastructure.' Journal of Statistical Software_, *94*(15), 1-64. doi: 10.18637/jss.v094.i15 (URL: https://doi.org/10.18637/jss.v094.i15).

Examples

```
## Simple linear program.
## maximize:   2 x_1 + 4 x_2 + 3 x_3
## subject to: 3 x_1 + 4 x_2 + 2 x_3 <= 60
##             2 x_1 +   x_2 +   x_3 <= 40
##               x_1 + 3 x_2 + 2 x_3 <= 80
##               x_1, x_2, x_3 are non-negative real numbers

LP <- OP( c(2, 4, 3),
          L_constraint(L = matrix(c(3, 2, 1, 4, 1, 3, 2, 2, 2), nrow = 3),
                       dir = c("<=", "<=", "<="),
                       rhs = c(60, 40, 80)),
          max = TRUE )
LP

## Simple quadratic program.
## minimize: - 5 x_2 + 1/2 (x_1^2 + x_2^2 + x_3^2)
## subject to: -4 x_1 - 3 x_2       >= -8
##              2 x_1 +   x_2       >=  2
##                    - 2 x_2 + x_3 >=  0

QP <- OP( Q_objective (Q = diag(1, 3), L = c(0, -5, 0)),
          L_constraint(L = matrix(c(-4,-3,0,2,1,0,0,-2,1),
                                  ncol = 3, byrow = TRUE),
                       dir = rep(">=", 3),
                       rhs = c(-8,2,0)) )
QP
```

`OP_signature` *Optimization Problem Signature*

Description

Takes an object of class `"OP"` (optimization problem) and returns the signature of the optimization problem.

Usage

```
OP_signature(x)
```

Arguments

`x`	an object of class `"OP"`

Value

A `data.frame` giving the signature of the the optimization problem.

`Q_constraint` *Quadratic Constraints*

Description

Quadratic constraints are typically of the form

$$\frac{1}{2}x^\top Q_i x + L_i x \leq rhs_i$$

where Q_i is the ith of m (sparse) matrices (all of dimension $n \times n$) giving the coefficients of the quadratic part of the equation. The $m \times n$ (sparse) matrix L holds the coefficients of the linear part of the equation and L_i refers to the ith row. The right hand side of the constraints is represented by the vector rhs.

Usage

```
Q_constraint(Q, L, dir, rhs, names = NULL)

## S3 method for class 'Q_constraint'
variable.names(object, ...)

as.Q_constraint(x)

is.Q_constraint(x)

## S3 method for class 'Q_constraint'
length(x)

## S3 method for class 'Q_constraint'
terms(x, ...)
```

Arguments

`Q`	a list of (sparse) matrices representing the quadratic part of each constraint.
`L`	a numeric vector of length n (a single constraint) or a matrix of dimension $m \times n$, where n is the number of objective variables and m is the number of constraints. Matrices can be of class `"simple_triplet_matrix"` to allow a sparse representation of constraints.
`dir`	a character vector with the directions of the constraints. Each element must be one of `"<="`, `">="` or `"=="`.
`rhs`	a numeric vector with the right hand side of the constraints.
`names`	an optional character vector giving the names of x (row/column names of Q, column names of A).
`object`	an R object.
`...`	further arguments passed to or from other methods (currently ignored).
`x`	an R object.

Value

an object of class "Q_constraint" which inherits from "constraint".

Author(s)

Stefan Theussl

Q_objective *Quadratic Objective Function*

Description

A quadratic objective function is typically of the form

$$\frac{1}{2}x^\top Qx + c^\top x$$

where Q is a (sparse) matrix defining the quadratic part of the function and c is a (sparse) vector of coefficients to the n defining the linear part.

Usage

```
Q_objective(Q, L = NULL, names = NULL)

## S3 method for class 'Q_objective'
terms(x, ...)

as.Q_objective(x)

## S3 method for class 'Q_objective'
variable.names(object, ...)
```

Arguments

Q	a $n \times n$ matrix with numeric entries representing the quadratic part of objective function. Sparse matrices of class "simple_triplet_matrix" can be supplied.
L	a numeric vector of length n, where n is the number of objective variables.
names	an optional character vector giving the names of x (row/column names of Q, column names of L).
x	an R object.
...	further arguments passed to or from other methods
object	an R object.

Value

an object of class "Q_objective" which inherits from "objective".

Author(s)

Stefan Theussl

`rbind.constraint` *Combine Constraints*

Description

Take a sequence of constraints (ROI objects) arguments and combine by rows, i.e., putting several constraints together.

Usage

```
## S3 method for class 'constraint'
rbind(..., use.names = FALSE, recursive = FALSE)
```

Arguments

`...`	constraints objects to be concatenated.
`use.names`	a logical if `FALSE` the names of the constraints are ignored when combining them, if `TRUE` the constraints are combined based on their `variable.names`.
`recursive`	a logical, if TRUE, rbind .

Details

The output type is determined from the highest type of the components in the hierarchy
`"L_constraint"` < `"Q_constraint"` < `"F_constraint"` and
`"L_constraint"` < `"C_constraint"`.

Value

an object of a class depending on the input which also inherits from `"constraint"`. See **Details**.

Author(s)

Stefan Theussl

`ROI_applicable_solvers` *Obtain Applicable Solvers*

Description

`ROI_applicable_solvers` takes as argument an optimization problem (object of class `'OP'`) and returns a vector giving the applicable solver. The set of applicable solver is restricted on the available solvers, which means if solver `"A"` and `"B"` would be applicable but a `ROI.plugin` is only installed for solver `"A"` only solver `"A"` would be listed as applicable solver.

Usage

```
ROI_applicable_solvers(op)
```

Arguments

`op`	an **ROI**-object of type `'OP'`.

Value

An character vector giving the applicable solver, for a certain optimization problem.

`ROI_available_solvers` *Available Solvers*

Description

ROI_available_solvers returns a data.frame of details corresponding to solvers currently available at one or more repositories. The current list of packages is downloaded over the Internet.

Usage

```
ROI_available_solvers(x = NULL, method = getOption("download.file.method"))
```

Arguments

`x`	an object used to select a method. It can be either an object of class `"OP"` or an object of class `"ROI_signature"` or `NULL`.
`method`	a character string giving the method to be used for downloading files. For more information see `download.file`.

Details

To get an overview about the available solvers `ROI_available_solvers()` can be used. If a signature or an object of class `"OP"` is provided **ROI** will only return the solvers applicable the optimization problem. Note since NLP solver are also applicable for LP and QP they will also be listed.

Value

a data.frame with one row per package and repository.

Examples

```
## Not run:
ROI_available_solvers()
op <- OP(1:2)
ROI_available_solvers(op)
ROI_available_solvers(OP_signature(op))

## End(Not run)
```

ROI_options *ROI Options*

Description

Allow the user to set and examine a variety of ROI options like the default solver or the function used to compute the gradients.

Usage

```
ROI_options(option, value)
```

Arguments

`option`	any options can be defined, using 'key, value' pairs. If 'value' is missing the current set value is returned for the given 'option'. If both are missing. all set options are returned.
`value`	the corresponding value to set for the given option.

ROI_plugin_add_status_code_to_db

Add Status Code to the Status Database

Description

Add a status code to the status database.

Usage

```
ROI_plugin_add_status_code_to_db(solver, code, symbol, message, roi_code = 1L)
```

Arguments

solver	a character string giving the name of the solver.
code	an integer giving the status code of the solver.
symbol	a character string giving the status symbol.
message	a character string used as status message.
roi_code	an integer giving the ROI status code, 1L for failure and 0L for success.

See Also

Other plugin functions: ROI_plugin_build_equality_constraints(), ROI_plugin_build_inequality_constraints(), ROI_plugin_canonicalize_solution(), ROI_plugin_get_solver_name(), ROI_plugin_make_signature(), ROI_plugin_register_solver_control(), ROI_plugin_register_solver_method(), ROI_plugin_solution_prim(), ROI_registered_solver_control()

Examples

```
## Not run:
solver <- "ecos"
ROI_plugin_add_status_code_to_db(solver, 0L, "ECOS_OPTIMAL", "Optimal solution found.", 0L)
ROI_plugin_add_status_code_to_db(solver, -7L, "ECOS_FATAL", "Unknown problem in solver.", 1L)
solver <- "glpk"
ROI_plugin_add_status_code_to_db(solver, 5L, "GLP_OPT", "Solution is optimal.", 0L)
ROI_plugin_add_status_code_to_db(solver, 1L, "GLP_UNDEF", "Solution is undefined.", 1L)

## End(Not run)
```

ROI_plugin_build_equality_constraints
Build Functional Equality Constraints

Description

There exist different forms of functional equality constraints, this function transforms the form used in **ROI** into the forms commonly used by R optimization solvers.

Usage

```
ROI_plugin_build_equality_constraints(x, type = c("eq_zero", "eq_rhs"))
```

Arguments

`x`	an object of type `"OP"`.
`type`	an character giving the type of the function to be returned, possible values are `"eq_zero"` or `"eq_rhs"`. For more information see Details.

Details

There are two types of equality constraints commonly used in R

1. `eq\_zero`: $h(x) = 0$ and
2. `eq\_rhs`: $h(x) = rhs$.

Value

Returns one function, which combines all the functional constraints.

Note

This function only intended for plugin authors.

See Also

Other plugin functions: `ROI_plugin_add_status_code_to_db()`, `ROI_plugin_build_inequality_constraints()`, `ROI_plugin_canonicalize_solution()`, `ROI_plugin_get_solver_name()`, `ROI_plugin_make_signature()`, `ROI_plugin_register_solver_control()`, `ROI_plugin_register_solver_method()`, `ROI_plugin_solution_prim()`, `ROI_registered_solver_control()`

ROI_plugin_build_inequality_constraints
Build Functional Inequality Constraints

Description

There exist different forms of functional inequality constraints, this function transforms the form used in **ROI** into the forms commonly used by R optimization solvers.

Usage

```
ROI_plugin_build_inequality_constraints(x, type = c("leq_zero", "geq_zero"))
```

Arguments

`x`	an object of type `"OP"`.
`type`	an character giving the type of the function to be returned, possible values are `"leq\_zero"` and `"geq\_zero"`. For more information see Details.

Details

There are three types of inequality constraints commonly used in R

1. `leq\_zero`: $h(x) \leq 0$ and
2. `geq\_zero`: $h(x) \geq 0$ and
3. `leq_geq\_rhs`: $lhs \geq h(x) \leq rhs$.

Value

Returns one function, which combines all the functional constraints.

Note

This function only intended for plugin authors.

See Also

Other plugin functions: `ROI_plugin_add_status_code_to_db()`, `ROI_plugin_build_equality_constraints()`, `ROI_plugin_canonicalize_solution()`, `ROI_plugin_get_solver_name()`, `ROI_plugin_make_signature()`, `ROI_plugin_register_solver_control()`, `ROI_plugin_register_solver_method()`, `ROI_plugin_solution_prim()`, `ROI_registered_solver_control()`

ROI_plugin_canonicalize_solution
Canonicalize Solution

Description

Transform the solution to a standardized form.

Usage

```
ROI_plugin_canonicalize_solution(
  solution,
  optimum,
  status,
  solver,
  message = NULL,
  ...
)
```

Arguments

solution	a numeric or integer vector giving the solution of the optimization problem.
optimum	a numeric giving the optimal value.
status	an integer giving the status code (exit flag).
solver	a character string giving the name of the solver.
message	an optional R object giving the original solver message.
...	further arguments to be stored in the solution object.

Value

an object of class "OP_solution".

See Also

Other plugin functions: ROI_plugin_add_status_code_to_db(), ROI_plugin_build_equality_constraints(), ROI_plugin_build_inequality_constraints(), ROI_plugin_get_solver_name(), ROI_plugin_make_signature(), ROI_plugin_register_solver_control(), ROI_plugin_register_solver_method(), ROI_plugin_solution_prim(), ROI_registered_solver_control()

`ROI_plugin_get_solver_name` *Get Solver Name*

Description

Get the name of the solver plugin.

Usage

```
ROI_plugin_get_solver_name(pkgname)
```

Arguments

`pkgname`	a string giving the package name.

Value

Returns the name of the solver as character.

See Also

Other plugin functions: `ROI_plugin_add_status_code_to_db()`, `ROI_plugin_build_equality_constraints()`, `ROI_plugin_build_inequality_constraints()`, `ROI_plugin_canonicalize_solution()`, `ROI_plugin_make_signatu` `ROI_plugin_register_solver_control()`, `ROI_plugin_register_solver_method()`, `ROI_plugin_solution_prim()`, `ROI_registered_solver_control()`

`ROI_plugin_make_signature` *Make Signatures*

Description

Create a solver signature, the solver signatures are used to indicate which problem types can be solved by a given solver.

Usage

```
ROI_plugin_make_signature(...)
```

Arguments

`...`	signature definitions

Value

an object of class `"ROI_signature"` (inheriting from data.frame) with the supported signatures.

See Also

Other plugin functions: ROI_plugin_add_status_code_to_db(), ROI_plugin_build_equality_constraints(), ROI_plugin_build_inequality_constraints(), ROI_plugin_canonicalize_solution(), ROI_plugin_get_solver_r ROI_plugin_register_solver_control(), ROI_plugin_register_solver_method(), ROI_plugin_solution_prim(), ROI_registered_solver_control()

Examples

```
## ROI_make_LP_signatures
lp_signature <- ROI_plugin_make_signature( objective = "L",
                                           constraints = "L",
                                           types = c("C"),
                                           bounds = c("X", "V"),
                                           cones = c("X"),
                                           maximum = c(TRUE, FALSE) )
```

ROI_plugin_register_reader_writer

Register Reader / Writer Method

Description

Register a new reader / writer method to be used with `read.io` / `write.io`.

Usage

```
ROI_plugin_register_reader(type, solver, method)

ROI_plugin_register_writer(type, solver, signature, method)
```

Arguments

`type`	a character giving the type of the file (e.g. "mps_free", "mps_fixed", "lp_cplex", "lp_lpsolve", ...).
`solver`	a character giving the name of the plugin (e.g. "lpsolve").
`method`	a function registered as reader / writer method.
`signature`	a data.frame giving the signature of the optimization problems which can be read or written by the registered method.

Details

- **File Types**
- **Method**

Value

NULL on success

See Also

Other input output: ROI_read(), ROI_registered_reader(), ROI_registered_writer(), ROI_write()

ROI_plugin_register_reformulation
Register Reformulation Method

Description

Register a new reformulation method to be used with ROI_reformulate.

Usage

```
ROI_plugin_register_reformulation(
  from,
  to,
  method_name,
  method,
  description = "",
  cite = "",
  author = ""
)
```

Arguments

from	a data.frame with the supported signatures.
to	a data.frame with the supported signatures.
method_name	a character string giving the name of the method.
method	a function registered as solver method.
description	a optional character string giving a description of what the reformulation does.
cite	a optional character string indicating a reference, such as the name of a book.
author	a optional character string giving the name of the author.

Value

TRUE on success

See Also

Other reformulate functions: ROI_reformulate(), ROI_registered_reformulations()

ROI_plugin_register_solver_control

Register Solver Controls

Description

Register a new solver control argument.

Usage

```
ROI_plugin_register_solver_control(solver, args, roi_control = "X")
```

Arguments

solver	a character string giving the solver name.
args	a character vector specifying with the supported signatures.
roi_control	a character vector specifying the corresponding ROI control argument.

Value

TRUE on success

See Also

Other plugin functions: ROI_plugin_add_status_code_to_db(), ROI_plugin_build_equality_constraints(), ROI_plugin_build_inequality_constraints(), ROI_plugin_canonicalize_solution(), ROI_plugin_get_solver_n ROI_plugin_make_signature(), ROI_plugin_register_solver_method(), ROI_plugin_solution_prim(), ROI_registered_solver_control()

ROI_plugin_register_solver_method

Register Solver Method

Description

Register a new solver method.

Usage

```
ROI_plugin_register_solver_method(signatures, solver, method, plugin = solver)
```

Arguments

`signatures`	a data.frame with the supported signatures.
`solver`	a character string giving the solver name.
`method`	a function registered as solver method.
`plugin`	a character string giving the plgug-in name.

Value

TRUE on success

See Also

Other plugin functions: `ROI_plugin_add_status_code_to_db()`, `ROI_plugin_build_equality_constraints()`, `ROI_plugin_build_inequality_constraints()`, `ROI_plugin_canonicalize_solution()`, `ROI_plugin_get_solver_`
`ROI_plugin_make_signature()`, `ROI_plugin_register_solver_control()`, `ROI_plugin_solution_prim()`, `ROI_registered_solver_control()`

`ROI_plugin_solution_prim`

Extract solution from the solver.

Description

Generic getter functions used by the function `solution`. These functions can be used to write a solver specific getter function.

Usage

```
ROI_plugin_solution_prim(x, force = FALSE)

## S3 method for class 'OP_solution'
ROI_plugin_solution_prim(x, force = FALSE)

## S3 method for class 'OP_solution_set'
ROI_plugin_solution_prim(x, force = FALSE)

ROI_plugin_solution_dual(x)

ROI_plugin_solution_aux(x)

ROI_plugin_solution_psd(x)

ROI_plugin_solution_msg(x)

ROI_plugin_solution_status_code(x)
```

```
ROI_plugin_solution_status(x)

ROI_plugin_solution_objval(x, force = FALSE)
```

Arguments

x	an R object inheriting from solution or solutions.
force	a logical to control the return value in the case that the status code is equal to 1 (i.e. something went wrong). By default force is FALSE and a solution is only provided if the status code is equal to 0 (i.e. success). If force is TRUE **ROI** ignores the status code and also returns solutions where the solver signaled an issue.

Value

the corresponding solution/s.

See Also

Other plugin functions: ROI_plugin_add_status_code_to_db(), ROI_plugin_build_equality_constraints(), ROI_plugin_build_inequality_constraints(), ROI_plugin_canonicalize_solution(), ROI_plugin_get_solver_
ROI_plugin_make_signature(), ROI_plugin_register_solver_control(), ROI_plugin_register_solver_method(
ROI_registered_solver_control()

ROI_read *Read Optimization Problems*

Description

Reads an optimization problem from various file formats and returns an optimization problem of class "OP".

Usage

```
ROI_read(file, type, solver = NULL, ...)
```

Arguments

file	a character giving the name of the file the optimization problem is to be read from.
type	a character giving the type of the file (e.g. "mps_free", "mps_fixed", "lp_cplex", "lp_lpsolve", ...).
solver	an optional character giving the name of the plugin (e.g. "lpsolve").
...	further arguments passed on to the read method.

Value

x an optimization problem of class "OP".

See Also

Other input output: ROI_plugin_register_reader_writer, ROI_registered_reader(), ROI_registered_writer(), ROI_write()

ROI_reformulate   *Reformulate a Optimization Problem*

Description

Register a new reformulation method.

Usage

```
ROI_reformulate(x, to, method = NULL)
```

Arguments

x	an object of class 'OP' giving the optimization problem.
to	a data.frame with the supported signatures.
method	a character string giving the name of the method.

Details

Currently **ROI** provides two reformulation methods.

1. bqp_to_lp transforms binary quadratic problems to linear mixed integer problems.
2. qp_to_socp transforms quadratic problems with linear constraints to second-order cone problems.

Value

the reformulated optimization problem.

See Also

Other reformulate functions: ROI_plugin_register_reformulation(), ROI_registered_reformulations()

Examples

```
## Example from
## Boros, Endre, and Peter L. Hammer. "Pseudo-boolean optimization."
## Discrete applied mathematics 123, no. 1 (2002): 155-225.

## minimize: 3 x y + y z - x - 4 y - z + 6

Q <- rbind(c(0, 3, 0),
           c(3, 0, 1),
           c(0, 1, 0))
```

```
L <- c(-1, -4, -1)
x <- OP(objective = Q_objective(Q = Q, L = L), types = rep("B", 3))

## reformulate into a mixed integer linear problem
milp <- ROI_reformulate(x, "lp")

## reformulate into a second-order cone problem
socp <- ROI_reformulate(x, "socp")
```

ROI_registered_reader *List Registered Reader*

Description

Retrieve meta information about the registered reader

Usage

```
ROI_registered_reader(type = NULL)
```

Arguments

type	an optional character giving the type of the file (e.g. "mps_free", "mps_fixed", "lp_cplex", "lp_lpsolve", ...).

Value

x a data.frame containing information about the registered readers.

See Also

Other input output: ROI_plugin_register_reader_writer, ROI_read(), ROI_registered_writer(), ROI_write()

Examples

```
ROI_registered_reader()
ROI_registered_reader("mps_fixed")
```

ROI_registered_reformulations
Registered Reformulations

Description

Retrieve meta information about the registered reformulations.

Usage

```
ROI_registered_reformulations()
```

Value

a data.frame giving some information about the registered reformulation methods.

See Also

Other reformulate functions: ROI_plugin_register_reformulation(), ROI_reformulate()

Examples

```
ROI_registered_reformulations()
```

ROI_registered_solvers
Solver Tools

Description

Retrieve the names of installed or registered solvers.

Usage

```
ROI_registered_solvers(...)

ROI_installed_solvers(...)
```

Arguments

...	arguments passed on to installed.packages.

Details

Whereas ROI_installed_solvers() may lists the names of installed solvers that do not necessarily work, ROI_registered_solvers() lists all solvers that can be used to solve optimization problems.

Value

a named character vector.

Author(s)

Stefan Theussl

ROI_registered_solver_control *Registered Solver Controls*

Description

Retrieve the registered solver control arguments.

Usage

```
ROI_registered_solver_control(solver)
```

Arguments

`solver`	a character string giving the solver name.

Value

a `data.frame` giving the control arguments.

See Also

Other plugin functions: `ROI_plugin_add_status_code_to_db()`, `ROI_plugin_build_equality_constraints()`, `ROI_plugin_build_inequality_constraints()`, `ROI_plugin_canonicalize_solution()`, `ROI_plugin_get_solver_`
`ROI_plugin_make_signature()`, `ROI_plugin_register_solver_control()`, `ROI_plugin_register_solver_method(`
`ROI_plugin_solution_prim()`

ROI_registered_writer *Write Optimization Problems*

Description

Write an optimization problem to file.

Usage

```
ROI_registered_writer(signature = NULL)
```

Arguments

signature	an optimization problem of class "OP".

See Also

Other input output: ROI_plugin_register_reader_writer, ROI_read(), ROI_registered_reader(), ROI_write()

Examples

```
ROI_registered_writer()
op <- OP(1:2)
ROI_registered_writer(OP_signature(op))
```

ROI_require_solver *Require Solver*

Description

Loads the specified solver and registers it in an internal data base. A request to load an already loaded solver has no effect.

Usage

```
ROI_require_solver(solver, warn = 0)
```

Arguments

solver	a character string giving the solver name.
warn	an integer giving if the warn level. For warn = -1 the warning is ignored. For warn = 0 the warning is stored and printed later. For warn = 1 the warning is printed immediately. For warn = 2 the warning is turned into an error. Default is warn = 0.

Value

Returns TRUE on success otherwise FALSE.

`ROI_solve` *Solve an Optimization Problem*

Description

Solve a given optimization problem. This function uses the given solver (or searches for an appropriate solver) to solve the supplied optimization problem.

Usage

```
ROI_solve(x, solver, control = list(), ...)
```

Arguments

`x`	an optimization problem of class `"OP"`.
`solver`	a character vector specifying the solver to use. If missing, then the default solver returned by `ROI_options` is used.
`control`	a list with additional control parameters for the solver. This is solver specific so please consult the corresponding documentation.
`...`	a list of control parameters (overruling those specified in `control`).

Value

a list containing the solution and a message from the solver.

- solutionthe vector of optimal coefficients
- objvalthe value of the objective function at the optimum
- statusa list giving the status code and message form the solver. The status code is 0 on success (no error occurred) 1 otherwise.
- messagea list giving the original message provided by the solver.

Author(s)

Stefan Theussl

References

Theussl S, Schwendinger F, Hornik K (2020). 'ROI: An Extensible R Optimization Infrastructure.' Journal of Statistical Software_, *94*(15), 1-64. doi: 10.18637/jss.v094.i15 (URL: https://doi.org/10.18637/jss.v094.i15).

Examples

```
## Rosenbrock Banana Function
## -----------------------------------------
## objective
f <- function(x) {
   return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
}
## gradient
g <- function(x) {
   return( c( -400 * x[1] * (x[2] - x[1] * x[1]) - 2 * (1 - x[1]),
             200 * (x[2] - x[1] * x[1])) )
}
## bounds
b <- V_bound(li = 1:2, ui = 1:2, lb = c(-3, -3), ub = c(3, 3))
op <- OP( objective = F_objective(f, n = 2L, G = g),
          bounds = b )
res <- ROI_solve( op, solver = "nlminb", control = list(start = c( -1.2, 1 )) )
solution( res )
## Portfolio optimization - minimum variance
## -----------------------------------------
## get monthly returns of 30 US stocks
data( US30 )
r <- na.omit( US30 )
## objective function to minimize
obj <- Q_objective( 2*cov(r) )
## full investment constraint
full_invest <- L_constraint( rep(1, ncol(US30)), "==", 1 )
## create optimization problem / long-only
op <- OP( objective = obj, constraints = full_invest )
## solve the problem - only works if a QP solver is registered
## Not run:
res <- ROI_solve( op )
res
sol <- solution( res )
names( sol ) <- colnames( US30 )
round( sol[ which(sol > 1/10^6) ], 3 )

## End(Not run)
```

ROI_solver_signature *Obtain Solver Signature*

Description

Obtain the signature of a registered solver.

Usage

```
ROI_solver_signature(solver)
```

Arguments

solver	a character string giving the name of the solver.

Value

the solver signature if the specified solver is registered NULL otherwise.

Examples

```
ROI_solver_signature("nlminb")
```

ROI_write *Write Optimization Problems*

Description

Write an optimization problem to file.

Usage

```
ROI_write(x, file, type, solver = NULL, ...)
```

Arguments

x	an optimization problem of class "OP".
file	a character giving the name of the file the optimization problem is to be written.
type	a character giving the type of the file (e.g. "freemps", "mps_fixed", "lp_cplex", "lp_lpsolve", ...).
solver	an optional character giving the name of the plugin (e.g. "lpsolve").
...	further arguments passed on to the write method.

See Also

Other input output: ROI_plugin_register_reader_writer, ROI_read(), ROI_registered_reader(), ROI_registered_writer()

solution — *Extract Solution*

Description

The solution can be accessed via the method 'solution'.

Usage

```
solution(
  x,
  type = c("primal", "dual", "aux", "psd", "msg", "objval", "status", "status_code"),
  force = FALSE,
  ...
)
```

Arguments

`x`	an object of type 'OP_solution' or 'OP_solution_set'.
`type`	a character giving the name of the solution to be extracted.
`force`	a logical to control the return value in the case that the status code is equal to 1 (i.e. something went wrong). By default force is `FALSE` and a solution is only provided if the status code is equal to 0 (i.e. success). If force is `TRUE` **ROI** ignores the status code and also returns solutions where the solver signaled an issue.
`...`	further arguments passed to or from other methods.

Value

the extracted solution.

types (Set/Get) — *Types - Accessor and Mutator Functions*

Description

The types of a given optimization problem (OP) can be accessed or mutated via the method 'types'.

Usage

```
types(x)

types(x) <- value
```

Arguments

`x`	an object used to select the method.
`value`	an R object.

Value

a character vector.

Author(s)

Stefan Theussl

Examples

```
## minimize: x + 2 y
## subject to: x + y >= 1
## x, y >= 0    x, y are integer
x <- OP(objective = 1:2, constraints = L_constraint(c(1, 1), ">=", 1))
types(x) <- c("I", "I")
types(x)
```

`US30` *Monthly return data for 30 of the largest US stocks*

Description

This dataset contains the historical monthly returns of 30 of the largest US stocks from 1999-01-29 to 2013-12-31. This data is dividend adjusted based on the CRSP methodology.

Format

A matrix with 30 columns (representing stocks) and 180 rows (months).

Details

The selected stocks reflect the DJ 30 Industrial Average Index members as of 2013-09-20 (downloaded from `https://www.quandl.com` which was acquired by https://data.nasdaq.com/).

The data source is Quandl. Data flagged as "WIKI" in their database is public domain.

Source

https://data.nasdaq.com/

vech — *Half-Vectorization*

Description

The utility function vech performs a half-vectorization on the given matrices.

Usage

```
vech(...)
```

Arguments

...	one or more matrices to be half-vectorized.

Value

a matrix

V_bound — *Objective Variable Bounds*

Description

Constructs a variable bounds object.

Usage

```
V_bound(li, ui, lb, ub, nobj, ld = 0, ud = Inf, names = NULL)

as.V_bound(x)

is.V_bound(x)
```

Arguments

li	an integer vector specifying the indices of non-standard (i.e., values != 0) lower bounds.
ui	an integer vector specifying the indices of non-standard (i.e., values != Inf) upper bounds.
lb	a numeric vector with lower bounds.
ub	a numeric vector with upper bounds.
nobj	an integer representing the number of objective variables
ld	a numeric giving lower default bound.
ud	a numeric giving upper default bound.
names	a character vector giving the names of the bounds.
x	object to be coerced or tested.

Details

This function returns a sparse representation of objective variable bounds.

Value

An S3 object of class "V_bound" containing lower and upper bounds of the objective variables.

Examples

```
V_bound(li=1:3, lb=rep.int(-Inf, 3))
V_bound(li=c(1, 5, 10), ui=13, lb=rep.int(-Inf, 3), ub=100, nobj=20)
```

Index