Package ‘ompr’

June 11, 2018

Type Package

Title Model and Solve Mixed Integer Linear Programs

Version 0.8.0

Description Model mixed integer linear programs in an algebraic way directly in R. The model is solver-independent and thus offers the possibility to solve a model with different solvers. It currently only supports linear constraints and objective functions. See the ‘ompr’ website <https://dirkschumacher.github.io/ompr> for more information, documentation and examples.

License GPL-3

LazyData TRUE

RoxygenNote 6.0.1

URL https://github.com/dirkschumacher/ompr

BugReports https://github.com/dirkschumacher/ompr/issues

Depends R (>= 3.2.0)

Imports lazyeval, progress, rlang (>= 0.2.0), methods, data.table, Rcpp (>= 0.12.12), Matrix

Suggests magrittr, testthat

ByteCompile Yes

LinkingTo Rcpp

NeedsCompilation yes

Author Dirk Schumacher [aut, cre]

Maintainer Dirk Schumacher <mail@dirk-schumacher.net>

Repository CRAN

Date/Publication 2018-06-11 18:53:53 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>*, LinearVariableCollection, numeric-method</td>
<td>3</td>
</tr>
<tr>
<td>*, LinearVariableSum, numeric-method</td>
<td>3</td>
</tr>
<tr>
<td>*, numeric, LinearVariableCollection-method</td>
<td>4</td>
</tr>
<tr>
<td>*, numeric, LinearVariableSum-method</td>
<td>4</td>
</tr>
<tr>
<td>+, LinearVariableCollection, LinearVariableCollection-method</td>
<td>5</td>
</tr>
<tr>
<td>+, LinearVariableCollection, LinearVariableSum-method</td>
<td>5</td>
</tr>
<tr>
<td>+, LinearVariableCollection, missing-method</td>
<td>6</td>
</tr>
<tr>
<td>+, LinearVariableCollection, numeric-method</td>
<td>6</td>
</tr>
<tr>
<td>+, LinearVariableSum, LinearVariableCollection-method</td>
<td>7</td>
</tr>
<tr>
<td>+, LinearVariableSum, LinearVariableSum-method</td>
<td>7</td>
</tr>
<tr>
<td>+, LinearVariableSum, missing-method</td>
<td>8</td>
</tr>
<tr>
<td>+, LinearVariableSum, numeric-method</td>
<td>8</td>
</tr>
<tr>
<td>+, numeric, LinearVariableCollection-method</td>
<td>9</td>
</tr>
<tr>
<td>+, numeric, LinearVariableSum-method</td>
<td>9</td>
</tr>
<tr>
<td>-, LinearVariableCollection, LinearVariableCollection-method</td>
<td>10</td>
</tr>
<tr>
<td>-, LinearVariableCollection, LinearVariableSum-method</td>
<td>10</td>
</tr>
<tr>
<td>-, LinearVariableCollection, missing-method</td>
<td>11</td>
</tr>
<tr>
<td>-, LinearVariableCollection, numeric-method</td>
<td>11</td>
</tr>
<tr>
<td>-, LinearVariableSum, LinearVariableCollection-method</td>
<td>12</td>
</tr>
<tr>
<td>-, LinearVariableSum, LinearVariableSum-method</td>
<td>12</td>
</tr>
<tr>
<td>-, LinearVariableSum, missing-method</td>
<td>13</td>
</tr>
<tr>
<td>-, LinearVariableSum, numeric-method</td>
<td>13</td>
</tr>
<tr>
<td>//, LinearVariableCollection, numeric-method</td>
<td>14</td>
</tr>
<tr>
<td>//, LinearVariableSum, numeric-method</td>
<td>14</td>
</tr>
<tr>
<td>add_constraint</td>
<td>15</td>
</tr>
<tr>
<td>add_variable</td>
<td>16</td>
</tr>
<tr>
<td>as_colwise</td>
<td>17</td>
</tr>
<tr>
<td>colwise</td>
<td>18</td>
</tr>
<tr>
<td>extract_constraints</td>
<td>19</td>
</tr>
<tr>
<td>get_column_duals</td>
<td>19</td>
</tr>
<tr>
<td>get_row_duals</td>
<td>20</td>
</tr>
<tr>
<td>get_solution</td>
<td>21</td>
</tr>
<tr>
<td>LinearVariable-class</td>
<td>22</td>
</tr>
<tr>
<td>LinearVariableCollection-class</td>
<td>22</td>
</tr>
<tr>
<td>LinearVariableSum-class</td>
<td>22</td>
</tr>
<tr>
<td>MILPModel</td>
<td>23</td>
</tr>
<tr>
<td>MIPModel</td>
<td>23</td>
</tr>
<tr>
<td>nconstraints</td>
<td>23</td>
</tr>
<tr>
<td>new_solution</td>
<td>24</td>
</tr>
<tr>
<td>nvars</td>
<td>25</td>
</tr>
<tr>
<td>objective_function</td>
<td>25</td>
</tr>
<tr>
<td>objective_value</td>
<td>26</td>
</tr>
<tr>
<td>ompr</td>
<td>26</td>
</tr>
<tr>
<td>set_bounds</td>
<td>27</td>
</tr>
</tbody>
</table>
set_objective ............................................................. 27
solver_status .......................................................... 28
solve_model ............................................................ 29
sum_expr ................................................................. 29
variable_bounds ....................................................... 30
variable_keys ......................................................... 30
variable_types ....................................................... 31
[,LinearVariableCollection,ANY,ANY,missing-method ....................... 32

Index 33

*,LinearVariableCollection,numeric-method

Multiply

Description

Multiplies the coefficients rowwise with a given numeric vector. If the numeric vector is a ‘linear\_transposed\_vector’, it will multiply the vector with each variable per row.

Usage

```r
## S4 method for signature 'LinearVariableCollection,numeric'
e1 * e2
```

Arguments

e1 an object of type 'LinearVariableCollection'
e2 a numeric vector

*,LinearVariableSum,numeric-method

Multiply

Description

It will multiply the numeric vector with both the constant and the variable in 'LinearVariableSum'

Usage

```r
## S4 method for signature 'LinearVariableSum,numeric'
e1 * e2
```

Arguments

e1 an object of type 'LinearVariableSum'
e2 a numeric vector
Description

Equivalent to 'e2 * e1'

Usage

```r
## S4 method for signature 'numeric,LinearVariableCollection'
e1 * e2
```

Arguments

e1                a numeric value
e2                an object of type 'LinearVariableCollection'

Description

It will multiply the numeric vector with both the constant and the variable in 'LinearVariableSum'

Usage

```r
## S4 method for signature 'numeric,LinearVariableSum'
e1 * e2
```

Arguments

e1                a numeric vector
e2                an object of type 'LinearVariableSum'
### Description

Adds two variables together. Same values for variable, row and col will be added. Everything else merged.

### Usage

```r
## S4 method for signature 'LinearVariableCollection,LinearVariableCollection'
e1 + e2
```

### Arguments

- `e1`: an object of type 'LinearVariableCollection'
- `e2`: an object of type 'LinearVariableCollection'

### Description

Equivalent to `e2 + e1`

### Usage

```r
## S4 method for signature 'LinearVariableCollection,LinearVariableSum'
e1 + e2
```

### Arguments

- `e1`: an object of type 'LinearVariableCollection'
- `e2`: an object of type 'LinearVariableSum'
Unary Plus

Description

Equivalent to 'e1'

Usage

```r
## S4 method for signature 'LinearVariableCollection,missing'
e1 + e2
```

Arguments

- `e1`: an object of type 'LinearVariableCollection'
- `e2`: a missing value

Plus

Description

Adds a constant numeric vector to a variable. The constant needs to be a vector of length 1.

Usage

```r
## S4 method for signature 'LinearVariableCollection,numeric'
e1 + e2
```

Arguments

- `e1`: an object of type 'LinearVariableCollection'
- `e2`: a numeric vector without NAs
Description

Adds the variables in the rhs to the variables in the lhs and returns another 'LinearVariableSum'.

Usage

```r
## S4 method for signature 'LinearVariableSum,LinearVariableCollection'
e1 + e2
```

Arguments

- `e1`: an object of type 'LinearVariableSum'
- `e2`: an object of type 'LinearVariableCollection'

Value

Returns an object of type 'LinearVariableSum'

---

Description

Add two object of 'LinearVariableSum'. I.e. variables + constants

Usage

```r
## S4 method for signature 'LinearVariableSum,LinearVariableSum'
e1 + e2
```

Arguments

- `e1`: an object of type 'LinearVariableSum'
- `e2`: an object of type 'LinearVariableSum'

Value

Returns an object of type 'LinearVariableSum'
Unary Plus

Description

Equivalent to `e1`

Usage

```r
## S4 method for signature 'LinearVariableSum,missing'
e1 + e2
```

Arguments

- `e1`: an object of type 'LinearVariableSum'
- `e2`: a missing value

Plus

Description

Adds a constant (rhs) to constant slot of the lhs object.

Usage

```r
## S4 method for signature 'LinearVariableSum,numerical'
e1 + e2
```

Arguments

- `e1`: an object of type 'LinearVariableSum'
- `e2`: a numeric vector

Value

an object of type 'LinearVariableSum'
Description

Equivalent to `e2 + e1`

Usage

## S4 method for signature 'numeric,LinearVariableCollection'
```
e1 + e2
```

Arguments

- `e1`: a numeric value
- `e2`: an object of type `LinearVariableCollection`

Description

Equivalent to `e2 + e1`

Usage

## S4 method for signature 'numeric,LinearVariableSum'
```
e1 + e2
```

Arguments

- `e1`: a numeric vector
- `e2`: an object of type `LinearVariableSum`
### Description
Equivalent to ‘e1 + -1 * e2’

### Usage
```r
## S4 method for signature 'LinearVariableCollection,LinearVariableCollection'
e1 - e2
```

### Arguments
- **e1**: an object of type 'LinearVariableCollection'
- **e2**: an object of type 'LinearVariableCollection'

### Description
Equivalent to ‘-1 * (e2 - e1)’

### Usage
```r
## S4 method for signature 'LinearVariableCollection,LinearVariableSum'
e1 - e2
```

### Arguments
- **e1**: an object of type 'LinearVariableCollection'
- **e2**: an object of type 'LinearVariableSum'
-,LinearVariableCollection,missing-method

Unary Minus

Description

Equivalent to 'e1 * (-1)'

Usage

```r
## S4 method for signature 'LinearVariableCollection,missing'
- e1
```

Arguments

- `e1`: an object of type 'LinearVariableCollection'
- `e2`: a missing value

-,LinearVariableCollection,numeric-method

Plus

Description

Equivalent to 'e1 + -1 * e2'

Usage

```r
## S4 method for signature 'LinearVariableCollection,numeric'
- e1
```

Arguments

- `e1`: an object of type 'LinearVariableCollection'
- `e2`: a numeric value
Description

Equivalent to 'e1 + -1 * e2'

Usage

## S4 method for signature 'LinearVariableSum,LinearVariableCollection'

```r
e1 - e2
```

Arguments

e1 an object of type 'LinearVariableSum'
e2 an object of type 'LinearVariableCollection'

Value

Returns an object of type 'LinearVariableSum'
Unary Minus

Description

Equivalent to \( e1 \times (-1) \)

Usage

```r
## S4 method for signature 'LinearVariableSum,missing'
e1 - e2
```

Arguments

- `e1`: an object of type 'LinearVariableSum'
- `e2`: a missing value

Minus

Description

Equivalent to \( e1 + -1 \times e2 \)

Usage

```r
## S4 method for signature 'LinearVariableSum,numerical'
e1 - e2
```

Arguments

- `e1`: an object of type 'LinearVariableSum'
- `e2`: a numeric vector
Description

Equivalent to '(-1 * e2) - (-1 * e1)'

Usage

```r
## S4 method for signature 'numeric,LinearVariableCollection'
e1 - e2
```

Arguments

e1 a numeric value
e2 an object of type 'LinearVariableCollection'

---

Description

Equivalent to '(-1 * e2) - (-1 * e1)'

Usage

```r
## S4 method for signature 'numeric,LinearVariableSum'
e1 - e2
```

Arguments

e1 a numeric vector
e2 an object of type 'LinearVariableSum'
Description

Equivalent to ‘e1 * (1 / e2)’

Usage

```r
## S4 method for signature 'LinearVariableCollection,numeric'
e1 / e2
```

Arguments

- **e1**: an object of type 'LinearVariableCollection'
- **e2**: a numeric value
add_constraint

Add a constraint

Description

Add one or more constraints to the model using quantifiers.

Usage

```r
add_constraint(.model, .constraint_expr, ..., .show_progress_bar = TRUE)

add_constraint(.model, .constraint_expr, ..., .dots, .show_progress_bar = TRUE)
```

Arguments

- `.model` the model
- `.constraint_expr` the constraint. Must be a linear (in)equality with operator "<=", "==" or ">=".
- `...` quantifiers for the indexed variables. For all combinations of bound variables a new constraint is created. In addition you can add filter expressions
- `.show_progress_bar` displays a progressbar when adding multiple constraints
- `.dots` Used to work around non-standard evaluation.

Value

a Model with new constraints added

Examples

```r
library(magrittr)
MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  add_constraint(x[i] >= 1, i = 1:5) # creates 5 constraints
```
add_variable

Add a variable to the model

Description

A variable can either be a name or an indexed name. See examples.

Usage

```r
add_variable(.model, .variable, ..., type = "continuous", lb = -Inf, ub = Inf)
add_variable(.model, .variable, ..., type = "continuous", lb = -Inf, ub = Inf, .dots)
```

Arguments

- `.model` the model
- `.variable` the variable name/definition
- `...` quantifiers for the indexed variable. Including filters
- `type` must be either continuous, integer or binary
- `lb` the lower bound of the variable
- `ub` the upper bound of the variable
- `.dots` Used to work around non-standard evaluation.

Examples

```r
library(magrittr)
MIPModel() %>%
  add_variable(x) %>% # creates 1 variable named x
  add_variable(y[i], i = 1:10, i %% 2 == 0, type = "binary") # creates 4 variables
```

as_colwise

As_colwise

Description

Convert lists or vectors to colwise semantic.

Usage

```r
as_colwise(x)
```
Arguments

- `x` a list of numeric vectors or a numeric vector

Description

This function should be used if you want to expand a variable across columns and not rows. When passing a vector of indexes to MILPModel variable, it creates a new row for each vector element. With colwise you can create columns instead. Please see the examples below.

Usage

```r
colwise(...)```

Arguments

```r
... create a colwise vector
```

Examples

```r
## Not run:
# vectors create matrix rows
# x[1, 1]
# x[2, 1]
# x[3, 1]
# x[1:3, 1]

# colwise() creates columns per row
# 1 * x[1, 1] + 2 * x[1, 2] + 3 * x[1, 3]
colwise(1, 2, 3) * x[1, colwise(1, 2, 3)]

# you can also combine the two
# x[1, 1]
# x[2, 1] + x[2, 2]
# x[3, 1] + x[3, 2] + x[3, 2]
x[1:3, colwise(1, 1:2, 1:3)]

## End(Not run)```
**extract_constraints**

**Description**

Extract the constraint matrix, the right hand side and the sense from a model

**Usage**

```r
extract_constraints(model)
```

**Arguments**

- **model**
  - the model

**Value**

A list with three named elements. `matrix` the (sparse) constraint matrix from the Matrix package. `rhs` is the right hand side vector in the order of the matrix. `sense` is a vector of the constraint senses

**Examples**

```r
library(magrittr)
model <- MIPModel() %>%
  add_variable(x[i], i = 1:3) %>%
  add_variable(y[i], i = 1:3) %>%
  add_constraint(x[i] + y[i] <= 1, i = 1:3)
extract_constraints(model)
```

---

**get_column_duals**

**Description**

Gets the column duals of a solution

**Usage**

```r
get_column_duals(solution)
```

**Arguments**

- **solution**
  - a solution
get_row_duals

Value

Either a numeric vector with one element per column or ‘NA_real_’.

Examples

## Not run:

```r
result <- MIPModel()
add_variable(x[i], i = 1:5)
add_variable(y[i, j], i = 1:5, j = 1:5)
add_constraint(x[i] >= 1, i = 1:5)
set_bounds(x[i], lb = 3, i = 1:3)
set_objective(sum_expr(i * x[i], i = 1:5))
solve_model(with_ROI("glpk"))

get_column_duals(result)
```

## End(Not run)

---

get_row_duals Gets the row duals of a solution

Description

Gets the row duals of a solution

Usage

`get_row_duals(solution)`

Arguments

- `solution` a solution

Value

Either a numeric vector with one element per row or ‘NA_real_’.

Examples

## Not run:

```r
result <- MIPModel()
add_variable(x[i], i = 1:5)
add_variable(y[i, j], i = 1:5, j = 1:5)
add_constraint(x[i] >= 1, i = 1:5)
set_bounds(x[i], lb = 3, i = 1:3)
set_objective(sum_expr(i * x[i], i = 1:5))
solve_model(with_ROI("glpk"))

get_column_duals(result)
```

## End(Not run)
**get_solution**

get_row_duals(result)

## End(Not run)

---

**get_solution**  
*Get variable values from a solution*

**Description**

Get variable values from a solution

**Usage**

```r
get_solution(solution, expr, type = "primal")
get_solution(solution, expr, type)
```

**Arguments**

- `solution`: the solution object
- `expr`: a variable expression. You can partially bind indexes.
- `type`: optional, either "primal" or "dual". The default value is "primal". If "primal" it returns the primal solution, otherwise the column duals. Especially the dual values depend on the solver. If no duals are calculated, the function stops with an error message.

**Value**

a data.frame. One row for each variable instance and a column for each index. Unless it is a single variable, then it returns a single number.

**Examples**

```r
## Not run:
library(magrittr)
result <- MIPModel()
  add_variable(x[i], i = 1:5)
  add_variable(y[i, j], i = 1:5, j = 1:5)
  add_constraint(x[i] >= 1, i = 1:5)
  set_bounds(x[i], lb = 3, i = 1:3)
  set_objective(0)
  solve_model(with_ROI("glpk"))
solution <- get_solution(result, x[i])
solution2 <- get_solution(result, y[i, 1])
solution3 <- get_solution(result, y[i, j])
duals <- get_solution(result, x[i], type = "duals")
## End(Not run)
```
LinearVariable-class

An S4 class that represents a single variable

Description

An S4 class that represents a single variable

Slots

variable a linear variable collection with just one index '1'

LinearVariableCollection-class

An S4 class that represents a collection of variables

Description

An S4 class that represents a collection of variables

Slots

variables a data frame hold the variable coefficients. One line for each variable, row and column.
index_mapping a function that takes a variable name as character and returns a mapping table that maps column ids to variable indexes.

LinearVariableSum-class

Holds a sum of a constant and a linear variable collection

Description

Holds a sum of a constant and a linear variable collection

Slots

constant a numeric vector
variables a variable collection
**MILPModel**  
*Create a new MILP Model*

---

**Description**  
Create an an empty mixed-integer linear programming model that is about 1000 times faster than ‘MIPModel’. It will eventually replace the old ‘MIPModel’ backend for linear models.

**Usage**  
MILPModel()

**Details**  
Please only use it if you can deal with potential API changes in the future.

---

**MIPModel**  
*Create a new MIP Model*

---

**Description**  
Create a new MIP Model

**Usage**  
MIPModel()

---

**nconstraints**  
*Number of variables (rows) of the model*

---

**Description**  
Number of variables (rows) of the model

**Usage**  
nconstraints(model)

**Arguments**  
model the model
Value

An integer equal to the number of variables. A variable is here a column in the resulting constraint matrix.

Examples

library(magrittr)
model <- MIPSmodel() %>%
  add_variable(x) %>%
  add_variable(y[i], i = 1:10)
ncconstraints(model) # 11

new_solution

Create a new solution

Description

This function/class should only be used if you develop your own solver.

Usage

new_solution(model, objective_value, status, solution,
             solution_column_duals = function() NA_real_,
             solution_row_duals = function() NA_real_)

Arguments

model
  the optimization model that was solved
objective_value
  a numeric objective value
status
  the status of the solution
solution
  a named numeric vector containing the primal solution values
solution_column_duals
  A function without arguments that returns a numeric vector containing the column dual solution values. ‘NA_real_’, if no column duals are available/defined.
solution_row_duals
  A function without arguments that returns a numeric vector containing the column dual solution values. ‘NA_real_’, if no column duals are available/defined.
nvars

Number of variables of a model

Description

Number of variables of a model

Usage

nvars(model)

Arguments

model the model

Value

a list with three named elements. 'binary' => number of binary variables, 'integer' => number of integer variables, 'continuous' => number of continuous variables.

Examples

library(magrittr)
model <- MIPModel()
  add_variable(x[i], i = 1:10, type = "binary")
  add_variable(y[i], i = 1:5, type = "continuous")
  add_variable(z[i], i = 1:2, type = "integer")

nvars(model)


objective_function

Extract the objective function from a model

Description

Extract the objective function from a model

Usage

objective_function(model)

Arguments

model the model
Value

a list with two named elements, ‘solution’ and ‘constant’. ‘solution’ is a sparse vector from the Matrix package. ‘constant’ is a constant that needs to be added to get the final obj. value.

Examples

```r
library(magrittr)
model <- MIPModel()
  add_variable(x[i], i = 1:5)
  set_objective(sum_expr(i * x[i], i = 1:5) + 10)
objective_function(model)
```

```
objective_value  Extract the numerical objective value from a solution

Description

Extract the numerical objective value from a solution

Usage

objective_value(solution)

Arguments

solution a solution

Value

numeric single item vector

ompr A package to Model (Mixed) Integer Programs

Description

A package to model (mixed) integer programs. It provides an algebraic way to model mixed integer linear optimization problems directly in R. The model is solver-independent and thus offers the possibility to solve a model with different solvers. See the ompr website <https://dirkschumacher.github.io/ompr/> for more information, documentation and examples.
**set_bounds**

*Set the bounds of a variable*

**Description**

Change the lower and upper bounds of a named variable, indexed variable or a group of variables.

**Usage**

```
set_bounds(.model, .variable, ..., lb = NULL, ub = NULL)
set_bounds_(.model, .variable, ..., lb = NULL, ub = NULL, .dots)
```

**Arguments**

- `.model` the model
- `.variable` the variable name/definition
- `...` quantifiers for the indexed variable
- `lb` the lower bound of the variable
- `ub` the upper bound of the variable
- `.dots` Used to work around non-standard evaluation.

**Examples**

```r
library(magrittr)
MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  add_constraint(x[i] >= 1, i = 1:5) %>% # creates 5 constraints
  set_bounds(x[i], lb = 3, i = 1:3)
```

---

**set_objective**

*Set the model objective*

**Description**

Set the model objective

**Usage**

```
set_objective(model, expression, sense = c("max", "min"))
set_objective_(model, expression, sense = c("max", "min"))
```
Arguments

model  the model
expression  the linear objective as a sum of variables and constants
sense  the model sense. Must be either "max" or "min".

Value

a Model with a new objective function definition

Examples

library(magrittr)
MIPModel() %>%
  add_variable(x, lb = 2) %>%
  add_variable(y, lb = 40) %>%
  set_objective(x + y, sense = "min")

---

solver_status  Get the solver status from a solution

Description

Get the solver status from a solution

Usage

solver_status(solution)

Arguments

solution  a solution

Value

color character vector being either "infeasible", "optimal", "unbounded", "userlimit" or "error"
solve_model

Solve a model

Description
Solve a model

Usage
solve_model(model, solver)

Arguments

model the model

solver a function mapping a model to a solution

Value
solver(model)

sum_expr
Construct a sum expression

Description
This function helps to create dynamic sum expression based on external variables. Should only be used within other `ompr` functions.

Usage
sum_expr(expr, ...)

Arguments
expr an expression that can be expanded to a sum
...
bind variables in expr using dots. See examples.

Value
the expanded sum as an AST

See Also
add_constraint
set_objective
Examples

# create a sum from x_1 to x_10
sum_expr(x[i], i = 1:10)
# create a sum from x_2 to x_10 with even indexes
sum_expr(x[i], i = 1:10, i %% 2 == 0)

variable_bounds

Variable lower and upper bounds of a model

Description

Variable lower and upper bounds of a model

Usage

variable_bounds(model)

Arguments

model the model

Value

a list with two components 'lower' and 'upper' each having a numeric vector of bounds. One for each variable.

Examples

library(magrittr)
model <- MIPModel()
  add_variable(x, type = "binary")
  add_variable(y, type = "continuous", lb = 2)
  add_variable(z, type = "integer", ub = 3)
variable_bounds(model)

variable_keys

Get all unique names of the model variables

Description

Get all unique names of the model variables

Usage

variable_keys(model)
variable_types

Arguments

model the model

Value

a character vector ordered in the same way as the constraint matrix columns and objective vector

Examples

library(magrittr)
model <- MIPModel()
add_variable(x[i], i = 1:3)
variable_keys(model)

variable_types Variable types of a model

Description

One component for each variable in the correct order

Usage

variable_types(model)

Arguments

model the model

Value

a factor with levels binary, continuous, integer

Examples

library(magrittr)
model <- MIPModel()
add_variable(x, type = "binary")
add_variable(y, type = "continuous")
add_variable(z, type = "integer")
variable_types(model)
### Subset model variables

**Description**

This creates a new variable collection as a subset of the previously defined indexed variable. A variable collection essentially is a data frame having values for rows and columns of the final model matrix.

**Usage**

```r
## S4 method for signature 'LinearVariableCollection,ANY,ANY,missing'

x[i, j, ...,
  drop = TRUE]
```

**Arguments**

- `x`: an object of type 'LinearVariableCollection'
- `i`: a numeric vector or a colwise vector/list
- `j`: a numeric vector or a colwise vector/list
- `...`: more a numeric vectors or a colwise vector/list
- `drop`: do not use this parameter

**Value**

a new object of type 'LinearVariableCollection'

**Examples**

```r
## Not run:

# vectors create matrix rows
# x[1, 1]
# x[2, 1]
# x[3, 1]

x[1:3, 1]

# colwise() creates columns per row
# 1 * x[1, 1] + 2 * x[1, 2] + 3 * x[1, 3]

colwise(1, 2, 3) * x[1, colwise(1, 2, 3)]

# you can also combine the two
# x[1, 1]
# x[2, 1] + x[2, 2]
# x[3, 1] + x[3, 2] + x[3, 2]

x[1:3, colwise(1, 1:2, 1:3)]

## End(Not run)
```
Index

Topic package
ompr, 26
*,LinearVariableCollection,numeric-method, 3
*,LinearVariableSum,numeric-method, 3
*,numeric,LinearVariableCollection-method, 4
*,numeric,LinearVariableSum-method, 4
+,LinearVariableCollection,LinearVariableCollection-method
add_constraint, 16, 29
add_constraint_(add_constraint), 16
add_variable, 17
add_variable_(add_variable), 17
as_colwise, 17
colwise, 18
extract_constraints, 19
get_column_duals, 19
get_row_duals, 20
get_solution, 21
get_solution_(get_solution), 21
LinearVariable-class, 22
LinearVariableCollection-class, 22
LinearVariableSum-class, 22
MILPModel, 23
MIPModel, 23
nconstraints, 23
new_solution, 24
nvars, 25
objective_function, 25
objective_value, 26
ompr, 26
ompr-package (ompr), 26
set_bounds, 27
-numeric,LinearVariableCollection-method, 14
-numeric,LinearVariableSum-method, 14
/,LinearVariableCollection,numeric-method, 15
/,LinearVariableSum,numeric-method, 15
[],LinearVariableCollection,ANY,ANY,missing-method, 32

+LinearVariableCollection,LinearVariableCollection-method
+LinearVariableCollection,LinearVariableSum-method
+LinearVariableCollection,missing-method, 6
+LinearVariableCollection,numeric-method, 6
+LinearVariableSum,LinearVariableCollection-method
+LinearVariableSum,LinearVariableSum-method
+LinearVariableSum,missing-method, 8
+LinearVariableSum,numeric-method, 8
+,numeric,LinearVariableCollection-method, 9
+,numeric,LinearVariableSum-method, 9
-LinearVariableCollection,LinearVariableCollection-method
-LinearVariableCollection,LinearVariableSum-method
-LinearVariableCollection,missing-method, 10
-LinearVariableCollection,numeric-method, 11
-LinearVariableSum,LinearVariableCollection-method
-LinearVariableSum,LinearVariableSum-method
-LinearVariableSum,missing-method, 13
-LinearVariableSum,numeric-method, 13

[LinearVariableCollection-any-any-missing-method, 32
add_constraint, 16, 29
add_constraint_(add_constraint), 16
add_variable, 17
add_variable_(add_variable), 17
as_colwise, 17
colwise, 18
extract_constraints, 19
get_column_duals, 19
get_row_duals, 20
get_solution, 21
get_solution_(get_solution), 21
LinearVariable-class, 22
LinearVariableCollection-class, 22
LinearVariableSum-class, 22
MILPModel, 23
MIPModel, 23
nconstraints, 23
new_solution, 24
nvars, 25
objective_function, 25
objective_value, 26
ompr, 26
ompr-package (ompr), 26
set_bounds, 27
set_bounds_ (set_bounds), 27
set_objective, 27, 29
set_objective_ (set_objective), 27
solve_model, 29
solver_status, 28
sum_expr, 29

variable_bounds, 30
variable_keys, 30
variable_types, 31