Package 'lpSolveAPI'

July 19, 2024

Version 5.5.2.0-17.12

Title R Interface to 'lp_solve' Version 5.5.2.0

Author lp_solve <https://lpsolve.sourceforge.net/> [aut],
Kjell Konis [aut],
Florian Schwendinger [aut, cre],
Kurt Hornik [ctb]

Maintainer Florian Schwendinger <FlorianSchwendinger@gmx.at>

Description The lpSolveAPI package provides an R interface to 'lp_solve',
a Mixed Integer Linear Programming (MILP) solver with support for pure
linear, (mixed) integer/binary, semi-continuous and special ordered sets
(SOS) models.

License LGPL-2

RoxygenNote 7.2.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-07-19 06:40:01 UTC

Contents

add.column *Add Column*

Description

Add a column to an lpSolve linear program model object.

Usage

```
add.column(lprec, x, indices)
```

Arguments

lprec	an lpSolve linear program model object.
x	a numeric vector containing the elements (only the nonzero elements if the indices argument is also provided) of the column to be added. The length of x must be equal to the number of constraints in lprec unless indices is provided.
indices	optional for sparse x. A numeric vector the same length as x of unique values from the set {0, ..., m} where m is the number of constraints in lprec; x[i] is entered into constraint indices[i] in the added column. The coefficients for the constraints not in indices are set to zero. In particular, index 0 is the objective function coefficient in the added column and is set to zero by default. This argument should be omitted when length(x) == m.

Details

This function adds an additional column to an lpSolve linear program model object. If multiple columns are to be added, performance can be improved by calling resize.lp before adding the columns.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

resize.lp

Examples

```
lps.model <- make.lp(4, 0)
add.column(lps.model, c(6,2,4,9))
add.column(lps.model, c(3,1,5), indices = c(1,2,4))
```

`add.constraint` *Add Constraint*

Description

Add a constraint to an lpSolve linear program model object.

Usage

```
add.constraint(lprec, xt, type = c("<=", "=", ">="), rhs, indices, lhs)
```

Arguments

`lprec`	an lpSolve linear program model object.
`xt`	a numeric vector containing the constraint coefficients (only the nonzero coefficients if `indices` is also given). The length of `xt` must be equal to the number of decision variables in `lprec` unless `indices` is provided.
`type`	a numeric or character value from the set {`1 = "<="`, `2 = ">="`, `3 = "="`} specifying the type of the constraint.
`rhs`	a single numeric value specifying the right-hand-side of the constraint.
`indices`	optional for sparse `xt`. A numeric vector the same length as `xt` of unique values from the set {1, ..., n} where n is the number of decision variables in `lprec`; `xt[i]` is entered into column `indices[i]` in the added constraint. The coefficients for the columns not in `indices` are set to zero. This argument should be omitted when `length(xt) == n`.
`lhs`	optional. A single numeric value specifying the left-hand-side of the constraint.

Details

Specifying the objective function before adding constraints will improve the performance of this function.

The use of this function should be avoided when possible. Building a model column-by-column rather than row-by-row will be on the order of 50 times faster (building the model - not solving the model).

Value

a `NULL` value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 4)
set.objfn(lps.model, rep(1, 4))
add.constraint(lps.model, c(6,2,4,9), "<=", 50)
add.constraint(lps.model, c(3,1,5), 2, 75, indices = c(1,2,4))
```

---

add.SOS *Add A Special Ordered Set Constraint*

Description

Add a Special Ordered Set (SOS) constraint to an lpSolve linear program model object.

Usage

```
add.SOS(lprec, name, type, priority, columns, weights)
```

Arguments

lprec	an lpSolve linear program model object.
name	a character string specifying a name for the SOS constraint.
type	a positive integer specifying the type of the SOS constraint.
priority	an integer specifying the priority of the SOS constraint.
columns	a numeric vector of unique values from the set {1, ..., n} (where n is the number of decision variables in lprec) indicating which decision variables belong to the special ordered set.
weights	a numeric vector the same length as columns specifying the variable weights.

Value

If the operation was successful: a single integer value containing the list index of the new special ordered set. A return value of 0 indicates an error.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

`delete.column` *Delete Column*

Description

Delete a column from an lpSolve linear program model object.

Usage

```
delete.column(lprec, columns)
```

Arguments

`lprec`	an lpSolve linear program model object.
`columns`	a numeric vector of unique values from the set `{1, ..., n}` (where `n` is the number of decision variables in `lprec`) specifying which columns should be deleted.

Value

a `NULL` value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(4, 0)

x <- c(6,2,4,9)
add.column(lps.model, x)

y <- c(3,1,5)
ind <- c(1,2,4)
add.column(lps.model, y, ind)

delete.column(lps.model, 1)
```

`delete.constraint` *Delete Constraint*

Description

Delete a constraint from an lpSolve linear program model object.

Usage

```
delete.constraint(lprec, constraints)
```

Arguments

`lprec`	an lpSolve linear program model object.
`constraints`	a numeric vector of unique values from the set `{1, ..., m}` (where `m` is the number of constraints in `lprec`) specifying which constraints should be deleted.

Value

a `NULL` value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 4)
set.objfn(lps.model, rep(1, 4))

xt <- c(6,2,4,9)
add.constraint(lps.model, xt, "<=", 50)

yt <- c(3,1,5)
ind <- c(1,2,4)
add.constraint(lps.model, yt, 2, 75, ind)

delete.constraint(lps.model, 1)
```

delete.lp *Delete Linear Program Model*

Description

Free all the resources used by an lpSolve linear program model object and set the value of the external pointer to NULL.

Usage

```
delete.lp(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Details

This function is used as a finalizer for lpSolve linear program model objects.

Value

lprec is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

dim.lpExtPtr *Dimension of an Object*

Description

Retrieve the dimension (number of rows and columns) from an lpSolve linear program model object.

Usage

```
## S3 method for class 'lpExtPtr'
dim(x)
## S3 replacement method for class 'lpExtPtr'
dim(x) <- value
```

Arguments

x	an lpSolve linear program model object.
value	assignment is not supported.

Details

Setting the number of rows/columns is not supported. See resize.lp.

Value

an integer vector of length two containing the number of rows and the number of columns in the lpSolve linear program model object.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

resize.lp

Examples

```
lps.model <- make.lp(4, 0)

x <- c(6,2,4,9)
add.column(lps.model, x)

y <- c(3,1,5)
ind <- c(1,2,4)
add.column(lps.model, y, ind)

dim(lps.model)
```

---

dimnames.lpExtPtr *Dimnames of an Object*

Description

Retrieve or set the dimnames in an lpSolve linear program model object.

Usage

```
## S3 method for class 'lpExtPtr'
dimnames(x)
## S3 replacement method for class 'lpExtPtr'
dimnames(x) <- value
```

Arguments

x	an lpSolve linear program model object.
value	a list containing two character vectors of lengths m and n specifying the row names and the column names for the lpSolve linear program model object. The number of constraints in x is denoted by m and the number of decision variables by n.

Value

a list of two character vectors containing the row names and the column names in the lpSolve linear program model object.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(4, 0)

x <- c(6,2,4,9)
add.column(lps.model, x)

y <- c(3,1,5)
ind <- c(1,2,4)
add.column(lps.model, y, ind)

dimnames(lps.model) <- list(c("alpha", "bravo", "charlie", "delta"),
                    c("whiskey", "tango"))
```

---

get.basis *Get Basis*

Description

Retrieve the basis from a solved lpSolve linear program model object.

Usage

```
get.basis(lprec, nonbasic = FALSE)
```

Arguments

lprec	an lpSolve linear program model object.
nonbasic	a logical value. If TRUE, the nonbasic variables are returned as well.

Value

an integer vector containing the indices of the basic (and nonbasic if requested) variables. If an error occurs (for instance when calling get.basis on a model that has not yet been solved) a NULL value is returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(3, 3)
## build and solve model ##
get.basis(lps.model)
```

---

get.bounds *Get Bounds*

Description

Retrieve the bounds on the decision variables from an lpSolve linear program model object.

Usage

```
get.bounds(lprec, columns = 1:n)
```

Arguments

lprec	an lpSolve linear program model object.
columns	a numeric vector of unique values from the set {1, ..., n} (where n is the number of decision variables in lprec) specifying the decision variables for which the bounds should be retrieved.

Value

a list with components `lower` and `upper`.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(3, 3)
get.bounds(lps.model)
```

`get.branch.mode` *Get Branch Mode*

Description

Retrieve the branch mode for one or more decision variables from an lpSolve linear program model object.

Usage

```
get.branch.mode(lprec, columns = 1:n, as.char = TRUE)
```

Arguments

`lprec`	an lpSolve linear program model object.
`columns`	a numeric vector of unique values from the set `{1, ..., n}` (where `n` is the number of decision variables in `lprec`) specifying for which decision variables the branch modes should be retrieved. If `NULL` branch modes are retrieved for all the decision variables.
`as.char`	a logical value. If `TRUE` the branch mode is returned as a character string, otherwise the integer code used directly in lpSolve is returned.

Value

either a character vector or an integer vector containing the branch modes for the decision variables specified in `columns`. The possibilities are: `0` = `"ceiling"`, `1` = `"floor"` and `2` = `"auto"`.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(3, 3)
get.branch.mode(lps.model)
```

`get.column` *Get Column*

Description

Retrieve a column from an lpSolve linear program model object.

Usage

```
get.column(lprec, column)
```

Arguments

`lprec`	an lpSolve linear program model object.
`column`	a single numeric value from the set `{1, ..., n}` (where `n` is the number of decision variables in `lprec`) specifying which column to retrieve.

Value

a list with elements `column` and `nzrow` such that `column[i]` contains the constraint coefficient in row `nzrow[i]`. Rows not present in `nzrow` have coefficient zero in `column`.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(4, 0)

x <- c(6,2,4,9)
add.column(lps.model, x)

y <- c(3,1,5)
ind <- c(1,2,4)
add.column(lps.model, y, ind)

get.column(lps.model, 2)
```

`get.constr.type` *Get Constraint Type*

Description

Retrieve constraint types from an lpSolve linear program model object.

Usage

```
get.constr.type(lprec, constraints = 1:m, as.char = TRUE)
```

Arguments

`lprec`	an lpSolve linear program model object.
`constraints`	a numeric vector of unique values from the set `{1, ..., m}` (where `m` is the number of constraints in `lprec`) specifying the constraints for which the types will be retrieved.
`as.char`	a logical value. If `TRUE` the constraint type is returned as a character string, otherwise the integer code used internally by lpSolve is returned.

Value

either a character vector or an integer vector containing the types of the constraints specified in `constraints`. The possibilities are: `0` = `"free"`, `1` = `"<="`, `2` = `">="` and `3` = `"="`.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)

xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 15)

xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 15)

xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)

get.constr.type(lps.model)
```

get.constr.value *Get Constraint Value*

Description

Retrieve constraint values from an lpSolve linear program model object.

Usage

```
get.constr.value(lprec, side = c("rhs", "lhs"), constraints = 1:m)
```

Arguments

lprec	an lpSolve linear program model object.
side	either "rhs" (right-hand-side) or "lhs" (left-hand-side) - which constraint values to retrieve.
constraints	a numeric vector of unique values from the set {1, ..., m} (where m is the number of constraints in lprec) specifying the constraints for which the values will be retrieved.

Value

a numeric vector containing the constraint values.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

get.constraints *Get Constraints*

Description

Retrieve the values of the constraints from a successfully solved lpSolve linear program model object.

Usage

```
get.constraints(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Value

a numeric vector containing the values of the constraints.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
get.constraints(lps.model)
```

---

get.dual.solution *Get Dual Solution*

Description

Retrieve the values of the dual variables (the reduced costs) from a successfully solved lpSolve linear program model object.

Usage

```
get.dual.solution(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Value

a numeric vector containing the values of the dual variables. If an error occurs (for instance lprec has not been successfully solved) a NULL value is returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
get.dual.solution(lps.model)
```

---

get.kind *Get Kind*

Description

Retrieve the kind of a decision variable from an lpSolve linear program model object.

Usage

```
get.kind(lprec, columns = 1:n)
```

Arguments

lprec	an lpSolve linear program model object.
columns	a numeric vector of unique values from the set {1, ..., n} (where n is the number of decision variables in lprec) specifying the columns for which the kind will be retrieved.

Details

Decision variables have both a type and a kind. The type is either real or integer and indicates the type of values the decision variable may take. The kind is one of {standard, semi-continuous, SOS}. Semi-continuous decision variables can take allowed values between their upper and lower bound as well as zero. Please see the link in the references for a discussion of special ordered set (SOS) constraints.

Value

a character vector containing the kind of each decision variable specified in `columns`.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

get.kind(lps.model)
```

---

get.mat *Get Matrix Element*

Description

Retrieve a single element from the matrix of constraints.

Usage

```
get.mat(lprec, i, j)
```

Arguments

`lprec`	an lpSolve linear program model object.
`i`	a single numeric value from the set {1, . . . , m} (where `m` is the number of constraints in `lprec`) specifying the row of the matrix element.
`j`	a single numeric value from the set {1, . . . , n} (where `n` is the number of decision variables in `lprec`) specifying the column of the matrix element.

Value

a single numeric value.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

get.mat(lps.model, 2, 2)
```

---

get.objective *Get Objective*

Description

Retrieve the value of the objective function from a successfully solved lpSolve linear program model object.

Usage

```
get.objective(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Value

a single numeric value containing the value of the objective function.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
get.objective(lps.model)
```

---

get.primal.solution *Get Primal Solution*

Description

Retrieve the values of the primal variables from a successfully solved lpSolve linear program model object.

Usage

```
get.primal.solution(lprec, orig = FALSE)
```

Arguments

lprec	an lpSolve linear program model object.
orig	a logical value. When presolve is active, the size of the lp may decrease during solve. By default, the answer to this reduced problem is returned. Set this argument to TRUE to retrieve the solution to the original lp.

Value

a numeric vector containing the values of the primal variables. If an error occurs (for instance lprec has not been successfully solved) a NULL value is returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
get.primal.solution(lps.model)
```

---

get.rhs *Get Right-Hand-Side*

Description

Retrieve right-hand-side values from an lpSolve linear program model object.

Usage

```
get.rhs(lprec, constraints = 1:m)
```

Arguments

`lprec`	an lpSolve linear program model object.
`constraints`	a numeric vector of unique values from the set `{1, ..., m}` specifying the constraints for which the right-hand-side values should be retrieved.

Value

a numeric vector containing the right-hand-side values specified by `constraints`.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

get.rhs(lps.model)
```

`get.sensitivity.obj` *Get Sensitivity: Objective*

Description

Retrieve the sensitivity of the objective function from a successfully solved lpSolve linear program model object.

Usage

```
get.sensitivity.obj(lprec)
```

Arguments

`lprec`	an lpSolve linear program model object.

Value

a list with components

`objfrom`	a numeric vector of length `n` (where `n` is the number of decision variables in `lprec`) containing the values of the lower limits of the objective function.
`objtill`	a numeric vector of length `n` (where `n` is the number of decision variables in `lprec`) containing the values of the upper limits of the objective function.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
get.sensitivity.obj(lps.model)
```

---

get.sensitivity.objex *Get Sensitivity: Objective Extended*

Description

Retrieve the sensitivity of the objective function from a successfully solved lpSolve linear program model object.

Usage

```
get.sensitivity.objex(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Value

a list with components

objfrom	a numeric vector of length n (where n is the number of decision variables in lprec) containing the values of the lower limits of the objective function.
objtill	a numeric vector of length n (where n is the number of decision variables in lprec) containing the values of the upper limits of the objective function.
objfromvalue	a numeric vector of length n (where n is the number of decision variables in lprec) containing the values of the variables at their lower limit. Only applicable when the value of the variable is 0 (rejected).
objtillvalue	not used in this version of lpSolve.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
get.sensitivity.objex(lps.model)
```

---

get.sensitivity.rhs *Get Sensitivity: Right-Hand-Side*

Description

Retrieve the sensitivity of the constraints from a successfully solved lpSolve linear program model object.

Usage

```
get.sensitivity.rhs(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Value

a list with components

duals	a numeric vector of length m+n (where m is the number of constraints in m and n is the number of decision variables in lprec) containing the values of the dual variables (reduced costs).
dualsfrom	a numeric vector of length m+n (where m is the number of constraints in m and n is the number of decision variables in lprec) containing the values of the lower limits on the dual variables.
dualstill	a numeric vector of length m+n (where m is the number of constraints in m and n is the number of decision variables in lprec) containing the values of the upper limits on the dual variables.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
get.sensitivity.rhs(lps.model)
```

---

get.solutioncount *Get Solution Count*

Description

Computes the number of equal solutions in a successfully solved lpSolve linear program model object. This is only valid if there are integer, semi-continuous or SOS variables in the model so that the branch-and-bound algorithm is used. This count gives the number of solutions with the same optimal objective value.

Usage

```
get.solutioncount(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Value

a single integer value giving the number of solutions attaining the optimal objective value.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

get.total.iter *Get Total Iterations*

Description

Retrieves the total number of iterations from a successfully solved lpSolve linear program model object.

Usage

```
get.total.iter(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Details

If lprec contains integer variables then this function returns the number of iterations to find a relaxed solution plus the number of iterations in the B&B process. If lprec contains no integer variables then this function returns the number of iterations to find a solution.

Value

a single integer value giving the total number of iterations.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

get.total.nodes *Get Total Nodes*

Description

Retrieves the total number of nodes processed in the branch-and-bound algorithm from a successfully solved lpSolve linear program model object.

Usage

```
get.total.nodes(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Details

The output of this function is only applicable for models containing integer variables.

Value

a single integer value giving the total number of nodes processed.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

---

get.type *Get Type*

Description

Retrieve the type of a decision variable from an lpSolve linear program model object.

Usage

```
get.type(lprec, columns = 1:n, as.char = TRUE)
```

Arguments

lprec	an lpSolve linear program model object.
columns	a numeric vector of unique values from the set {1, ..., n} (where n is the number of decision variables in lprec) specifying the columns for which the type will be retrieved.
as.char	a logical value. If TRUE the constraint type is returned as a character string, otherwise the integer code used internally by lpSolve is returned.

Details

The function set.type can be used to set a decision variable as binary. A binary decision variable is simply an integer decision with an upper bound of one and a lower bound of zero - hence this function will report the type as integer.

Value

either a character vector with elements from the set {"real", "integer"} indicating the type of each decision variable specified in columns or (if as.char = FALSE) a logical vector with TRUE elements corresponding to the integer decision variables specified in columns.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

set.type

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
set.type(lps.model, 2, "binary")
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.type(lps.model, 3, "integer")
set.objfn(lps.model, c(-3,-4,-3))

get.type(lps.model)
```

---

get.variables *Get Variables*

Description

Retrieve the values of the decision variables from a successfully solved lpSolve linear program model object.

Usage

```
get.variables(lprec)
```

Arguments

lprec	an lpSolve linear program model object.

Value

a numeric vector containing the values of the decision variables corresponding to the optimal solution.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
get.variables(lps.model)
```

---

guess.basis *Guess Basis*

Description

Attempt to find a feasible basis corresponding to a user provided feasible point.

Usage

```
guess.basis(lprec, guess)
```

Arguments

lprec	an lpSolve linear program model object.
guess	a numeric vector of length n (the number of decision variables in lprec) containing a feasible point.

Value

if successful, a numeric vector containing the indices of a starting basis. This vector is suitable for use with the set.basis function.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

set.basis

---

lp.control *lpSolve Control Parameters*

Description

Set control parameters in an lpSolve linear program model object.

Usage

```
lp.control(lprec, ..., reset = FALSE)
```

Arguments

lprec	an lpSolve linear program model object.
...	control arguments to bet set in lprec.
reset	a logical value. If TRUE all control parameters are reset to their default values.

Value

a list containing all of the control parameters as set internally in lprec.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

lp.control.options

lp.control.options *Solver Control Options*

Description

A description of the various control options that can be set in an lpSolve linear program model object using the lp.control function.

Control Options

anti.degen a character vector containing one or more of the following options. If any element is "none" then no anti-degeneracy handling is used.

"none":	No anti-degeneracy handling.
"fixedvars":	Check if there are equality slacks in the basis and try to drive them out in order to reduce chance of degene
"columncheck":	
"stalling":	
"numfailure":	
"lostfeas":	
"infeasible":	
"dynamic":	
"duringbb":	
"rhsperturb":	Perturbation of the working RHS at refactorization
"boundflip":	Limit bound flips that can sometimes contribute to degeneracy in some models.

The default is c("infeasible", "stalling", "fixedvars").

basis.crash a character string specifying the basis crash mode to use. When no basis crash is done the initial basis from which lpSolve starts is the basis containing all slack or artificial variables. When basis crash is enabled, a heuristic *crash procedure* is executed before the first simplex iteration to quickly choose a basis matrix that has fewer artificial variables. This procedure tends to reduce the number of iterations required by the solver. The choices for this option are given in the following table.

"none":	No basis crash.
"mostfeasible":	Most feasible basis.
"leastdegenerate":	Construct a basis that is in some sense the *least* degenerate.

The default is "none".

bb.depthlimit a single integer value specifying the maximum branch-and-bound depth. A positive value means that the depth limit is absolute. A negative value means a relative branch-and-bound depth limit. The *order* of an MIP problem is defined to be 2 times the number of binary variables plus the number of semi-continuous (SC) and special-ordered-sets (SOS) variables. A relative value of -x results in a maximum depth of x times the order of the MIP problem.

This control option only applies if there are integer, SC or SOS variables in the model (i.e., when the branch-and-bound algorithm is used). The branch-and-bound algorithm will not go deeper than this level. Limiting the depth speeds up the solving time but there is a chance that the solution obtained is sub-optimal. Be aware of this. Another possible consequence is that no solution will be found.

The default value is `-50`; a value of zero implies no limit to the depth.

bb.floorfirst a character string from among the following choices specifying which branch to take first in the branch-and-bound algorithm.

`"ceiling"`:	Take ceiling branch first.
`"floor"`:	Take floor branch first.
`"auto"`:	lpSolve decides which branch to take first.

The value of this option can influence solving times considerably. However, the real-world performance will be model dependent. The default is `"auto"`.

bb.rule a character vector specifying the branch-and-bound rule. The first element must be chosen from the following table.

`"first"`:	Select the lowest indexed non-integer column.
`"gap"`:	Selection based on the distance from the current bounds.
`"range"`:	Selection based on the largest current bound.
`"fraction"`:	Selection based on the largest fractional value.
`"pseudocost"`:	Simple, unweighted pseudo-cost of a variable.
`"pseudononint"`:	An extended pseudo-costing strategy based on minimizing the number of integer infeasibilities.
`"pseudoratio"`:	An extended pseudo-costing strategy based on maximizing the normal pseudo-cost divided by the numbe

Additional modes (if any) may be appended to augment the rule specified in the first element of `bb.rule`.

`"weightreverse"`:	Select by criterion minimum (worst), rather than by criterion maximum (best).
`"branchreverse"`:	When `bb.floorfirst` is `"auto"`, select the direction (lower/upper branch) opposite to that chosen by lp
`"greedy"`:	
`"pseudocost"`:	Toggle between weighting based on pseudocost or objective function value.
`"depthfirst"`:	Select the node that has been selected before the most number of times.
`"randomize"`:	Add a randomization factor to the score for all the node candidates.
`"gub"`:	This option is still in development and should not be used at this time.
`"dynamic"`:	When `"depthfirst"` is selected, switch it off once the first solution is found.
`"restart"`:	Regularly restart the pseudocost value calculations.
`"breadthfirst"`:	Select the node that has been selected the fewest number of times (or not at all).
`"autoorder"`:	Create an *optimal* branch-and-bound variable ordering. Can speed up branch-and-bound algorithm.
`"rcostfixing"`:	Do bound tightening during branch-and-bound based on the reduced cost information.
`"stronginit"`:	Initialize pseudo-costs by strong branching.

The value of this rule can influence solving times considerably. However, the real-world performance will be model dependent. The default value is `c("pseudononint", "greedy", "dynamic", "rcostfixing")`.

break.at.first a logical value. If TRUE then the branch-and-bound algorithm stops at the first solution found. The default (FALSE) is to continue until an optimal solution is found.

break.at.value a numeric value. The branch-and-bound algorithm stops if the objective function becomes better than this value. The default ($\pm$infinity) is to continue until an optimal value is found.

epslevel a character string providing a simplified way of specifying multiple tolerance thresholds in a *logically* consistent way. The following values are set: epsel, epsb, epsd, epspivot, epsint and mip.gap.

"tight":	Use tight tolerance values.
"medium":	Use medium tolerance values.
"loose":	Use loose tolerance values.
"baggy":	Use very loose tolerance values.

The default is "tight".

epsb a single positive numeric value specifying the tolerance used to determine whether a right-hand-side value should be considered zero. Rounding error in floating-point calculations may result in a loss of precision. A very small value (for example, 1e-99) could be the result of such errors and should be considered zero by the solver. If the absolute value of a right-hand-side value is less than epsb then it is treated as zero by the solver. The default value is 1.0e-10.

epsd a single positive numeric value specifying the tolerance used to determine whether a computed reduced cost should be considered zero. Rounding error in floating-point calculations may result in a loss of precision. A very small value (for example, 1e-99) could be the result of such errors and should be considered zero by the solver. If the absolute value of a computed reduced cost is less than epsd then it is treated as zero by the solver. The default value is 1.0e-9.

epsel a single positive numeric value specifying the tolerance used for rounding values to zero. Rounding error in floating-point calculations may result in a loss of precision. A very small value (for example, 1e-99) could be the result of such errors and should be considered zero by the solver. If the absolute value of a computed value is less than epsel then it is rounded to zero by the solver. The default value is 1.0e-12. This parameter is used in situations where none of epsint, epsb, epsd, epspivot norepsperturb apply.

epsint a single positive numeric value specifying the tolerance used to determine whether a floating-point number is an integer. This parameter only applies when there is at least one integer variable so that the branch and bound algorithm is used. Integer variables are internally stored as floating-point. A tolerance is therefore needed to determine whether a value should be considered an integer. If the absolute value of the variable minus the closest integer is less than epsint then it is considered an integer. The default value is 1.0e-7.

epsperturb a single positive numeric value specifying the perturbation scalar for degenerate problems. The default is 1.0e-5.

epspivot a single positive numeric value specifying the tolerance used to determine whether a pivot element is zero. Rounding error in floating-point calculations may result in a loss of precision. A very small value (for example, 1e-99) could be the result of such errors and should be considered zero by the solver. If the absolute value of a computed pivot element is less than epspivot then it is treated as zero by the solver. Pivots will be performed on elements smaller

(in absolute terms) than epspivot when no other larger pivot element can be found. The default value is 2.0e-7.

improve a character vector specifying the iterative improvement level. The possible values are given in the following table.

"none":	None.
"solution":	Running accuracy measurement of solved equations based on $Bx = r$ (primal simplex), remedy is refactoriz
"dualfeas":	Improve initial dual feasibility by bound flips (highly recommended).
"thetagap":	Low-cost accuracy monitoring in the dual, remedy is refactorization.
"bbsimplex":	By default there is a check for primal/dual feasibility at the optimum only for the relaxed problem, this also a

The default is c("dualfeas", "thetagap").

infinite a positive numeric value specifying the practical value for infinity. This value is used for very large numbers, for example the upper bound of a variable without an upper bound. The default is 1.0e30.

maxpivot a positive integer value specifying the maximum number of pivots between re-inversion of the matrix. For stability, lpSolve periodically re-inverts the matrix. However, the more often this is done, the slower the solver becomes. The default is 250.

mip.gap a numeric vector of length two specifying respectively the absolute and relative MIP gaps used in the branch-and-bound algorithm. This tolerance is the difference between the best-found solution yet and the current solution. If the difference is smaller than this tolerance then the solution (and all the sub-solutions) is rejected. This can result in faster solving times, but results in a solution which is not the perfect solution. The default is 1.0e-11.

negrange a nonpositive numeric value below which variables are split into negative and positive parts. The default is -1.0e6.

obj.in.bas a logical value specifying whether the objective function is stored in the matrix. The default is to store the objective function in the top row of the constraint matrix. If FALSE then the objective function is moved to separate storage. When the objective function is not stored in the basis the computation of reduced costs is somewhat slower. In the later versions of v5.5 there is the option to calculate reduced cost in the textbook way: completely independently of the basis.

pivoting a character vector specifying the pivot rule (the rule for selecting row and column entering/leaving) and mode. The first element of this vector must be one of the four pivot rules listed in the first table. Remaining elements (if any) specify modes that modify this rule. The rule/mode can influence solving times considerably.

"firstindex":	Select first.
"dantzig":	Select according to Dantzig.
"devex":	Devex pricing from Paula Harris.
"steepestedge":	Steepest edge.

"primalfallback":	When using the steepest edge rule, fall back to "devex" in the primal.
"multiple":	A preliminary implementation of the multiple pricing scheme. Attractive candidate columns from one i

"partial":	Enables partial pricing.
"adaptive":	Temporarily use an alternative strategy if cycling is detected.
"randomize":	Adds a small randomization effect to the selected pricer.
"autopartial":	Indicates automatic detection of segmented/staged/blocked models. It refers to partial pricing rather tha
"loopleft":	Scan entering/leaving columns left rather than right.
"loopalternate":	Scan entering/leaving columns alternating left/right.
"harristwopass":	Use Harris' primal pivot logic rather than the default.
"truenorminit":	Use true norms for Devex and steepest edge initializations.

The default is `c("devex", "adaptive")`.

presolve a character vector specifying presolve steps to be carried out before solving. Presolve looks at the model and tries to simplify it so that solving times are shorter. For example a constraint on only one variable is converted to a bound on this variable (and the constraint is deleted) - the model dimensions can change because of this. Both rows and columns can be deleted by the presolve. Also, note that the `"lindep"` presolve option can result in the deletion of rows (the linear dependent ones). The `get.constraints` function will then return only the values of the rows that are kept.

The presolve options are given in the following table. If any element of `presolve` is `"none"` then no presolving is done.

"none":	No presolve.
"rows":	Presolve rows.
"cols":	Presolve columns.
"lindep":	Eliminate linearly dependent rows.
"sos":	Convert constraints to special ordered sets (SOS), only SOS1 is handled.
"reducemip":	Constraints found redundant in phase 1 are deleted. This is no longer active since it is rarely effective and
"knapsack":	Simplification of knapsack-type constraints through the addition of an extra variable. This also helps boun
"elimeq2":	Direct substitution of one variable in 2-element equality constraints; this requires changes to the constraint
"impliedfree":	Identify implied free variables (releasing their explicit bounds).
"reducegcd":	Reduce (tighten) coefficients in integer models based on GCD argument.
"probefix":	Attempt to fix binary variables at one of their bounds.
"probereduce":	Attempt to reduce coefficients in binary models.
"rowdominate":	Identify and delete qualifying constraints that are dominated by others, also fixes variables at a bound.
"coldominate":	Delete variables (mainly binary) that are dominated by others (only one can be non-zero).
"mergerows":	Merges neighboring >= or <= constraints when the vectors are otherwise relatively identical into a single ra
"impliedslk":	Converts qualifying equalities to inequalities by converting a column singleton variable to a slack variable.
"colfixdual":	Variable fixing and removal based on the signs of the associated dual constraint.
"bounds":	Bound tightening based on full-row constraint information. This can assist in tightening the objective func
"duals":	Calculate duals.
"sensduals":	Calculate sensitivity if there are integer variables.

The default is `c("none")`.

scalelimit a numeric value specifying the relative scaling convergence criterion for the active scaling mode; the integer part specifies the maximum number of iterations. The default is `5`.

scaling a character vector specifying the scaling algorithm used and zero or more augmentations. The first element must be one of the scaling algorithms given in the following table.

"none":	No scaling (not advised).
"extreme":	Scale to convergence using largest absolute value.
"range":	Scale based on the simple numerical range.
"mean":	Numerical range-based scaling.
"geometric":	Geometric scaling.
"curtisreid":	Curtis-Reid scaling.

Additional elements (if any) from the following table can be included to augment the scaling algorithm.

"quadratic":	
"logarithmic":	Scale to convergence using logarithmic mean of all values.
"power2":	Power scaling.
"equilibrate":	Make sure that no scaled number is above 1.
"integers":	Scale integer variables.
"dynupdate":	Recompute scale factors when resolving the model.
"rowsonly":	Only scale rows.
"colsonly":	Only scale columns.

By default, lpSolve computes scale factors once for the original model. If a solve is done again (most probably after changing some data in the model), the scaling factors are not recomputed. Instead, the scale factors from the original model are used. This is not always desirable, especially if the data has changed considerably. Including "dynupdate" among the scale algorithm augmentations instructs lpSolve to recompute the scale factors each time solve is called. Note that the scaling done by "dynupdate" is incremental and the resulting scale factors are typically different from those computed from scratch.

The default is c("geometric", "equilibrate", "integers").

sense one of "max" or "min" specifying whether the model is a maximization or a minimization problem.

simplextype a character vector of length one or two composed of "primal" and "dual". If length two then the first element describes the simplex type used in phase 1 and the second element the simplex type used in phase 2. If length one then that simplex type is used for both phases. The default is c("dual", "primal").

timeout a positive integer value specifying the number of seconds after which a timeout will occur. If zero, then no timeout will occur.

verbose a character string controlling the severity of messages reported by lp_solve. The possible choices are given in the table below. All errors/warnings in lp_solve have a particular severiy: for example, specifying a wrong row or column index is considered a severe error. All messages equal to and below the set level are reported (in the console).

"neutral":	No reporting.
"critical":	Only critical messages are reported. Hard errors like instability, out of memory, etc.
"severe":	Only severe messages are reported. Errors.
"important":	Only important messages are reported. Warnings and Errors.
"normal":	Normal messages are reported.

"detailed": Detailed messages are reported. Like model size, continuing B&B improvements, etc.
"full": All messages are reported. Useful for debugging purposes and small models.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

lp.control

---

make.lp *Make LP*

Description

Create a new lpSolve linear program model object.

Usage

```
make.lp(nrow = 0, ncol = 0, verbose = "neutral")
```

Arguments

nrow	a nonnegative integer value specifying the number of constaints in the linear program.
ncol	a nonnegative integer value specifying the number of decision variables in the linear program.
verbose	a character string controlling the level of error reporting. The default value "neutral" is no error reporting. Use "normal" or "full" for more comprehensive error reporting. See the verbose entry in lp.control.options for a complete description of this argument and its possible values.

Value

an lpSolve linear program model object. Specifically an R external pointer with class lpExtPtr.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(4, 3)
```

`name.lp` *Name LP*

Description

Set or retrieve the name of an lpSolve linear program model object.

Usage

```
name.lp(lprec, name)
```

Arguments

`lprec`	an lpSolve linear program model object.
`name`	an optional character string giving a new name for `lprec`.

Details

If `name` is provided then this function sets the name of the lpSolve linear program model object. If `name` is missing then this function retrieves the name from `lprec`.

Value

there is no return value if the `name` argument is given. Otherwise a character string containing the name of the lpSolve linear program model object.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

name.lp(lps.model, "Simple LP")
name.lp(lps.model)
```

---

plot.lpExtPtr *lpExtPtr Plot Method*

Description

Plots the feasible set of a simple linear program with two decision variables. The decision variables must be real, nonnegative and must not have a finite upper bound. Only inequality constraints are supported.

Usage

```
## S3 method for class 'lpExtPtr'
plot(x, y, ...)
```

Arguments

x	an lpSolve linear program model object.
y	this argument is ignored.
...	additional arguments are ignored.

Author(s)

Kjell Konis <kjell.konis@me.com>

print.lpExtPtr *lpSolve Print Method*

Description

Display an lpSolve linear program model object in the console.

Usage

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

Arguments

x	an lpSolve linear program model object.
...	additional arguments are ignored.

Value

x is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

read.lp *Read LP*

Description

Read an lpSolve linear program model object from a file.

Usage

```
read.lp(filename, type = c("lp", "mps", "freemps"), verbose = "neutral",
        options)
```

Arguments

filename	a character string giving the name of the file which the linear programming model is to be read from.
type	the type of file provided in filename. If missing, read.lp will attempt to guess the file type by examining the file's suffix.
verbose	a character string controlling the level of error reporting. The default value "neutral" is no error reporting. Set to "normal" to enable error reporting or to "full" for a comprehensive parse log. See the verbose entry in lp.control.options for a complete description of this argument and its possible values.
options	a character vector of options for the (free)mps parser. Possible values are

free Use the free MPS format even when type = "mps". If not specified, the fixed MPS format is used. This option has no effect when type = "freemps" but will not result in an error or a warning.

ibm Interpret integer variables without bounds as binary variables per the original IBM standard. By default, lp_solve interprets integer variables without bounds as having no upper bound.

negobjconst Interpret the objective constant with an opposite sign. Some solvers interpret the objective constant as a value in the RHS and negate it when brought to the LHS. This option allows lp_solve to do this as well.

Value

an lpSolve linear program model object.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

---

resize.lp *Resize LP*

Description

Resize the data structures in an lpSolve linear program model object.

Usage

```
resize.lp(lprec, nrow, ncol)
```

Arguments

lprec	an lpSolve linear program model object.
nrow	a single nonnegative integer value specifying the new number of rows for the lpSolve linear program model object.
ncol	a single nonnegative integer value specifying the new number of rows for the lpSolve linear program model object.

Details

If the new size of the model is smaller than the size of the current model, the excess rows and/or columns are deleted. If the new size is larger, no change will be immediately apparent. However, the internal structures of lprec will have been adjusted to accommodate the larger model. Efficiency of model building can be improved by calling this function before adding additional columns (for example).

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(4, 0)
resize.lp(lps.model, 4, 2)

x <- c(6,2,4,9)
add.column(lps.model, x)

y <- c(3,1,5)
ind <- c(1,2,4)
add.column(lps.model, y, ind)
```

---

row.add.mode *Row Add Mode*

Description

Switch to row entry mode.

Usage

```
row.add.mode(lprec, state)
```

Arguments

lprec	an lpSolve linear program model object.
state	optional: either "on" or "off". This argument should be provided only to switch row entry mode on or off.

Details

The best way to build a linear program model in lpSolve is column by column, hence row entry mode is turned off by default. If the model must be built by adding constraints, the performance of the add.constraint function can be greatly improved by turning row entry mode on.

There are several caveats associated with row entry mode. First, only use this function on lpSolve linear program models created by make.lp. Do not use this function on models read from a file. Second, add the objective function before adding the constraints. Third, do not call any other API functions while in row entry mode; no other data matrix access is allowed. After adding all the contraints, turn row entry mode off. Once turned off, you cannot switch back to row entry mode.

Value

"on" if row entry mode is on in lprec; otherwise "off".

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

add.constraint

---

select.solution *Select Solution*

Description

Select which solution is returned by the lpSolve accessor methods.

Usage

```
select.solution(lprec, solution)
```

Arguments

lprec	an lpSolve linear program model object.
solution	optional. An integer between 1 and the number of optimal solutions to the model.

Details

When the branch and bound algorithm is used (i.e., when there are integer, semi-continuous or SOS variables in the model) there may be multiple optimal solutions.

Value

a single integer value: the number of optimal solutions.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

---

set.basis *Set Basis*

Description

Set the initial basis in an lpSolve linear program model object.

Usage

```
set.basis(lprec, basis, nonbasic = FALSE, default = FALSE)
```

Arguments

lprec	an lpSolve linear program model object.
basis	a numeric vector of unique values from the set {1, ..., (m+n)} (where m is the number of constraints and n is the number of decision variables) specifying the initial basis. The values may be positive or negative where a negative value indicates that the variable is at its lower bound and positive value indicates that the variable is at its upper bound. If nonbasic is FALSE then the n basic variables must be provided. If nonbasic is TRUE then the nonbasic variables must be provided as well.
nonbasic	a logical value. If TRUE the nonbasic variables must be included in basis as well.
default	a logical value. If TRUE the default (all slack variable) basis is used. In this case, the value of basis and nonbasic are ignored.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

---

set.bounds *Set Bounds*

Description

Set bounds on the decision variables in an lpSolve linear program model object.

Usage

```
set.bounds(lprec, lower = NULL, upper = NULL, columns = 1:n)
```

Arguments

lprec	an lpSolve linear program model object.
lower	a numeric vector of lower bounds to be set on the decision variables specified in columns. If NULL the lower bounds are not changed.
upper	a numeric vector of upper bounds to be set on the decision variables specified in columns. If NULL the upper bounds are not changed.
columns	a numeric vector of values from the set {1, ..., n} specifying the columns to have their bounds set. If NULL all columns are set.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 4)

set.bounds(lps.model, lower = rep(-1.0, 4))
set.bounds(lps.model, upper = 1:4)

set.bounds(lps.model, lower = rep(0.0, 4), upper = rep(1.0, 4))
```

`set.branch.mode` *Set Branch Mode*

Description

Specify which branch to take first in the branch-and-bound algorithm for decision variables in an lpSolve linear program model object.

Usage

```
set.branch.mode(lprec, columns, modes)
```

Arguments

`lprec`	an lpSolve linear program model object.
`columns`	a numeric vector containing values from the set `{1, ..., n}` (where `n` is the number of decision variables in `lprec`) specifying which columns to set the mode for.
`modes`	a character vector composed of the strings `{"ceiling", "floor", "auto", "default"}` giving the branch modes for the decision variables specified in `columns`. Please see the reference for a description of these terms.

Value

a `NULL` value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

set.branch.weights *Set Branch Weights*

Description

Set weights on the variables in an lpSolve linear program model object.

Usage

```
set.branch.weights(lprec, weights)
```

Arguments

lprec	an lpSolve linear program model object.
weights	a numeric vector with n elements (where n is the number of decision variables in lprec) containing the weights for the decision variables.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

set.column *Set Column*

Description

Set a column in an lpSolve linear program model object.

Usage

```
set.column(lprec, column, x, indices)
```

Arguments

lprec	an lpSolve linear program model object.
column	a single numeric value from the set {1, ..., n} specifying which column to set.
x	a numeric vector containing the elements (only the nonzero elements if indices is also given) to be used in the added column. The length of x must be equal to the number of constraints in lprec unless indices is provided.
indices	optional for sparse x. A numeric vector the same length as x of unique values from the set {0, ..., m} where m is the number of constraints in lprec; x[i] is set in constraint indices[i] in the specified column. The coefficients for the constraints not in indices are set to zero. In particular, index 0 is the objective function coefficient in the specified column and is set to zero by default. This argument should be omitted when length(x) == m.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(4, 2)
set.column(lps.model, 2, c(6,2,4,9))
set.column(lps.model, 1, c(3,1,5), indices = c(1,2,4))
```

---

set.constr.type *Set Constraint Type*

Description

Set constraint types in an lpSolve linear program model object.

Usage

```
set.constr.type(lprec, types, constraints = 1:m)
```

Arguments

lprec	an lpSolve linear program model object.
types	either a numeric vector or a character vector containing elements from the set {1 = "<=", 2 = ">=", 3 = "="} specifying the types of constraints. Additionally, the contraint type can be set to *free* uinsg the code 0.
constraints	a numeric vector of unique values from the set {1, ..., m} (where m is the number of constraints in lprec) specifying which constraints to set.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

resize.lp

Examples

```
lps.model <- make.lp(4, 2)

x <- c(6,2,4,9)
set.column(lps.model, 2, x)

y <- c(3,1,5)
ind <- c(1,2,4)
set.column(lps.model, 1, y, ind)

set.constr.type(lps.model, rep("<=", 4))
```

---

set.constr.value *Set Constraint Value*

Description

Set constraint values in an lpSolve linear program model object.

Usage

```
set.constr.value(lprec, rhs = NULL, lhs = NULL, constraints = 1:m)
```

Arguments

lprec	an lpSolve linear program model object.
rhs	a numeric vector the same length as constraints containing the right-hand-side values to be set. If NULL no right-hand-side values are set.
lhs	a numeric vector the same length as constraints containing the left-hand-side values to be set. If NULL no left-hand-side values are set.
constraints	a numeric vector of unique values from the set {1, ..., m} (where m is the number of constraints in lprec) specifying which constraints to set.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

resize.lp

---

set.mat *Set Matrix Element*

Description

Set the value of a single matrix element in an lpSolve linear program model object.

Usage

```
set.mat(lprec, i, j, value)
```

Arguments

lprec	an lpSolve linear program model object.
i	a single numeric value from the set {1, ..., m} (where m is the number of constraints in lprec) specifying the row of the matrix.
j	a single numeric value from the set {1, ..., n} (where n is the number of decision variables in lprec) specifying the column of the matrix.
value	a single numeric value.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

resize.lp

Examples

```
lps.model <- make.lp(4, 2)
x <- c(6,2,4,9)
set.column(lps.model, 2, x)
y <- c(3,1,5)
ind <- c(1,2,4)
set.column(lps.model, 1, y, ind)
set.constr.type(lps.model, rep("<=", 4))

set.mat(lps.model, 3, 2, 4.5)
```

---

set.objfn *Set Objective Function*

Description

Set the objective function in an lpSolve linear program model object.

Usage

```
set.objfn(lprec, obj, indices)
```

Arguments

lprec	an lpSolve linear program model object.
obj	a numeric vector of length n (where n is the number of decision variables in lprec) containing the coefficients of the objective function. Alternatively, if indices is also provided, a numeric vector of the same length as indices containing only the nonzero coefficients.
indices	optional for sparse obj. A numeric vector the same length as obj of unique values from the set {1, ..., n} where n is the number of decision variables in lprec; obj[i] is entered into column indices[i] in objective function. The coefficients for the columns not in indices are set to zero. This argument should be omitted when length(obj) == n.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

resize.lp

Examples

```
lps.model <- make.lp(2, 4)
set.objfn(lps.model, c(1,2,3,4))
set.objfn(lps.model, c(5,7,6), indices = c(1,2,4))
```

---

set.rhs *Set Right-Hand-Side*

Description

Set elements on the right-hand-side of an lpSolve linear program model object.

Usage

```
set.rhs(lprec, b, constraints = 1:m)
```

Arguments

lprec	an lpSolve linear program model object.
b	a numeric vector of length m (where m is the number of constraints in lprec)
constraints	a numeric vector containing unique elements from the set $\{1, \ldots, m\}$ (where m is the number of constraints in lprec) identifying the constraints for which to set the right-hand-side value.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(4, 2)
x <- c(6,2,4,9)
set.column(lps.model, 2, x)
y <- c(3,1,5)
ind <- c(1,2,4)
set.column(lps.model, 1, y, ind)

set.rhs(lps.model, c(10, 20, 40, 80))
```

---

set.row *Set Row*

Description

Set a column in an lpSolve linear program model object.

Usage

```
set.row(lprec, row, xt, indices)
```

Arguments

lprec	an lpSolve linear program model object.
row	a single numeric value from the set {1, ..., m} (where m is the number of constraints in lprec) specifying which column to set.
xt	a numeric vector containing the constraint coefficients (only the nonzero coefficients if indices is also given). The length of xt must be equal to the number of decision variables in lprec unless indices is provided.
indices	optional for sparse xt. A numeric vector the same length as xt of unique values from the set {1, ..., n} where n is the number of decision variables in lprec; xt[i] is set in column indices[i] in the specified row. The coefficients for the columns not in indices are set to zero. This argument should be omitted when length(xt) == n.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(2, 4)
set.row(lps.model, 2, c(6,2,4,9))
set.row(lps.model, 1, c(3,1,5), indices = c(1,2,4))
```

`set.semicont` *Set Semicontinuous*

Description

Set a decision variable as semicontinuous in an lpSolve linear program model object.

Usage

```
set.semicont(lprec, columns, sc = TRUE)
```

Arguments

`lprec`	an lpSolve linear program model object.
`columns`	a numeric vector of unique values from the set `{1, ..., n}` (where `n` is the number of decision variables in `lprec`) specifying which variables should be made semicontinuous.
`sc`	a logical value. If `FALSE` the decision variables specified in `columns` will have their `kind` set to `standard`.

Details

Decision variables have both a type and a kind. The type is either `real` or `integer` and indicates the type of values the decision variable may take. The kind is one of `{standard, semi-continuous, SOS}`. Semi-continuous decision variables can take allowed values between their upper and lower bound as well as zero. Please see the link in the references for further details.

Value

a `NULL` value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

set.type *Set Type*

Description

Set the type of a decision variable in an lpSolve linear program model object.

Usage

```
set.type(lprec, columns, type = c("integer", "binary", "real"))
```

Arguments

lprec	an lpSolve linear program model object.
columns	a numeric vector of unique values from the set {1, ..., n} (where n is the number of decision variables in lprec) specifying which variables are to have their type set.
type	either "integer", "binary" or "real". The decision variables in columns will have their type set to this value.

Details

A binary decision variable is simply an integer decision with an upper bound of one and a lower bound of zero. When type = "binary" the type of the decision variable will be set to "integer" and the bounds will be set to zero and one respectively.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

See Also

set.type

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
set.type(lps.model, 2, "binary")
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.type(lps.model, 3, "integer")
set.objfn(lps.model, c(-3,-4,-3))

get.type(lps.model)
```

`solve.lpExtPtr` *Solve a Linear Program*

Description

Attempt to compute the optimal solution of an lpSolve linear program model object.

Usage

```
## S3 method for class 'lpExtPtr'
solve(a, b, ...)
```

Arguments

a	an lpSolve linear program model object.
b	this argument is ignored.
...	additional arguments are ignored.

Value

a single integer value containing the status code.

Status Codes

0: "optimal solution found"
1: "the model is sub-optimal"
2: "the model is infeasible"
3: "the model is unbounded"
4: "the model is degenerate"
5: "numerical failure encountered"
6: "process aborted"
7: "timeout"
9: "the model was solved by presolve"

10: "the branch and bound routine failed"
11: "the branch and bound was stopped because of a break-at-first or break-at-value"
12: "a feasible branch and bound solution was found"
13: "no feasible branch and bound solution was found"

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Examples

```
lps.model <- make.lp(0, 3)
xt <- c(6,2,4)
add.constraint(lps.model, xt, "<=", 150)
xt <- c(1,1,6)
add.constraint(lps.model, xt, ">=", 0)
xt <- c(4,5,4)
add.constraint(lps.model, xt, "=", 40)
set.objfn(lps.model, c(-3,-4,-3))

solve(lps.model)
```

---

write.lp *Write Linear Program*

Description

Write an lpSolve linear program model object to a file.

Usage

```
write.lp(lprec, filename, type = c("lp", "mps", "freemps"),
         use.names = c(TRUE, TRUE))
```

Arguments

lprec	an lpSolve linear program model object.
filename	a character string containing the name of the output file.
type	either "lp", "mps" or "freemps". The type of file to output. Please see the reference for a description of these formats.

use.names — a logical vector of length two specifying repectively whether row and column names should be written into the file. If TRUE then the names are written into the file, otherwise lpSolve's internal names are used. This can be useful if the model names do not comply with the syntax of the chosen file type.

Value

a NULL value is invisibly returned.

Author(s)

Kjell Konis <kjell.konis@me.com>

References

https://lpsolve.sourceforge.net/5.5/index.htm

Index